Scalable Bandwidth Management in Software-Defined Networks

Prepared by:

Lindokuhle Zakithi Biyase

BYSLIN002

Department of Electrical Engineering University of Cape Town

Prepared for:

Alexandru Murgu

Department of Electrical Engineering University of Cape Town

November 2020

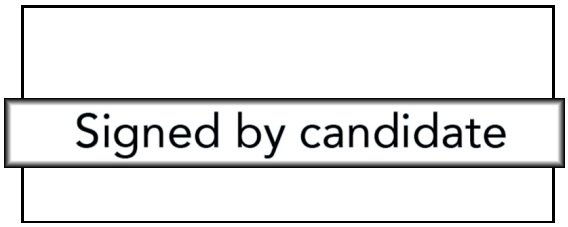
Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Masters in Engineering: Telecommunications Degree (**MEng Telecommunications)**

Key Words: SDN Controller, OpenFlow, Scalability, Mininet

Declaration

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor. This work is submitted for the Degree of Master of Engineering specializing in Telecommunications at the University of Cape Town. It has not been submitted to any other university for any other degree or examination.

Name: Lindokuhle Zakithi Biyase



Date: 20 November 2020

Abstract

There has been a growing demand to manage bandwidth as the network traffic increases. Network applications such as real time video streaming, voice over IP and video conferencing in IP networks has risen rapidly over the recently and is projected to continue in the future. These applications consume a lot of bandwidth resulting in increasing pressure on the networks. In dealing with such challenges, modern networks must be designed to be application sensitive and be able to offer Quality of Service (QoS) based on application requirements. Network paradigms such as Software Defined Networking (SDN) allows for direct network programmability to change the network behavior to suit the application needs in order to provide solutions to the challenge. In this dissertation, the objective is to research if SDN can provide scalable QoS requirements to a set of dynamic traffic flows.

Methods are implemented to attain scalable bandwidth management to provide high QoS with SDN. Differentiated Services Code Point (DSCP) values and DSCP remarking with Meters are used to implement high QoS requirements such that bandwidth guarantee is provided to a selected set of traffic flows. The theoretical methodology is implemented for achieving QoS, experiments are conducted to validate and illustrate that QoS can be implemented in SDN, but it is unable to implement High QoS due to the lack of implementation for Meters with DSCP remarking.

The research work presented in this dissertation aims at the identification and addressing the critical aspects related to the SDN based QoS provisioning using flow aggregation techniques. Several tests and demonstrations will be conducted by utilizing virtualization methods. The tests

are aimed at supporting the proposed ideas and aims at creating an improved understanding of the practical SDN use cases and the challenges that emerge in virtualized environments.

DiffServ Assured Forwarding is chosen as a QoS architecture for implementation. The bandwidth management scalability in SDN is proved based on throughput analysis by considering two conditions i.e 1) Per-flow QoS operation and 2) QoS by using DiffServ operation in the SDN environment with Ryu controller. The result shows that better performance QoS and bandwidth management is achieved using the QoS by DiffServ operation in SDN rather than the per-flow QoS operation.

Acknowledgements

I would convey my gratitude to the Creator of the Heavens and the Earth for having given me the courage and determination to pursue my studies. I wish to extend my appreciation to my supervisor, Dr Alexandru Murgu for continuous support, guidance and the advice he has given while working on the topic and for providing great insights on how to tackle the scalability. I pass my gratitude to my family and friends for the continuous support they have given me.

Glossary

API – Application Programming Interface

CLI - Command Line Interface

IP –Internet Protocol

ISP - Internet Service Provider

KPI - Key Performance Indicator

KPP - Key Performance Parameters

NBI – Northbound Interface

NNPP-Non-Network Performance Parameters

NNP – Non- Network Performance

NP – Network Performance

NPP- Network Performance Parameters

NSP - Network Service Provider

OF – OpenFlow

REST - Representational State Transfer (REST)

SBI - Southbound Interface

SDN – Software Defined Network

SDNC - SDN Controller

SLA – Service Level Agreement

UDP – User Datagram Protocol

The above abbreviations were gathered from different sources in the telecommunication spectrum, used to serve as a definition for this study and are specifically taken from IEEE standards supporting cognitive Radio and Networks, IEEE P1900.

Contents

List of Tables

1. Introduction

1.1 Problem Description

There has been a growing traction in the usage of services that requires a good quality of service (QoS) in IP networks such as video streaming, video conferencing online gaming etc. The increased usage of such high QoS applications is as a result of the increased computational capacity and display capabilities of user networking devices. The usage of video streaming applications is projected to contribute about 82% of the total IP traffic by the year 2020 [1].

The growing number of internet users presents many challenges such as:

- The inability for the network to self-regulate or self-grow in order to match with the growing demand.
- The network has a set finite capacity in terms of storage and data transmission rate across any two network nodes.
- Adaption from the user applications and advanced traffic control as the insufficient network resources may lead to network congestion.

The internet must always offer a good QoS and that requires innovative thinking and implementation by network engineers or administrators to ensure that the QoS is not degraded. It is essential for the network management team to consider innovative ways and strategies to scale the network up in order to cope with the growing traffic generated by the growing population.

For the new applications to satisfy their Service Level Agreement (SLA) requirements, the next generation networks must be application aware and allow and prioritize applications that have

highly sensitive requirements to negotiate with a network controller to reserve network resources such that the offered service meets its QoS requirements (bandwidth, jitter and delay).

When a user application has negotiated successfully with the network controller, network resources are granted and offered to a dedicated channel for that application. This is achieved through automatic configuration of flows and port priorities to satisfy the agreed network QoS and SLA. The next generation network technology architecture such as the Software Defined Networking (SDN) allows for such implementations and other innovations.

The decoupling of the control plane from the data plane abstracts the lower level functions into higher level services. This is advantageous for:

- Allowing the policy enforcement in the network.
- Allowing flexible network configurations.

The SDN architecture is based on a centralized controller and the distributed switching mechanism in the data plane. This brings scalability concerns as the controller has limited finite computing capacity and as a result, the controller can support a limited number of switching elements before it gets into what is known as performance bottleneck which can create a single point of failure.

1.2 Project Objectives

The objective of this dissertation is to research if SDN will be able to provide the required QoS to chosen hosts. The QoS should enable end-to-end bandwidth to guarantee between selected hosts.

In order to provide a good QoS the following features must be applied:

- Provide end-to-end bandwidth guarantee between selected hosts
- Prioritization of traffic depending on the required QoS parameter guarantee and thus protecting the given limits and constraints.
- Allowing excess traffic and offering good QoS when network resources are available.

1.3 Motivation

Communication networks are an essential component in today's society. They offer innovative services that meets the diverse connectivity needs of the different customers. The customers always expect an excellent service from the network that meets a specific connectivity service requirement for the services they consume, which are offered at competitive price rates.

In order to satisfy the service requirements, the ISPs must ensure that:

- The communication network operates at standard level such that the network services are delivered within acceptable quality standards while ensuring that the available network resources are utilized as effectively as possible.
- The Service Level Agreement is adhered to when offering services to end users.

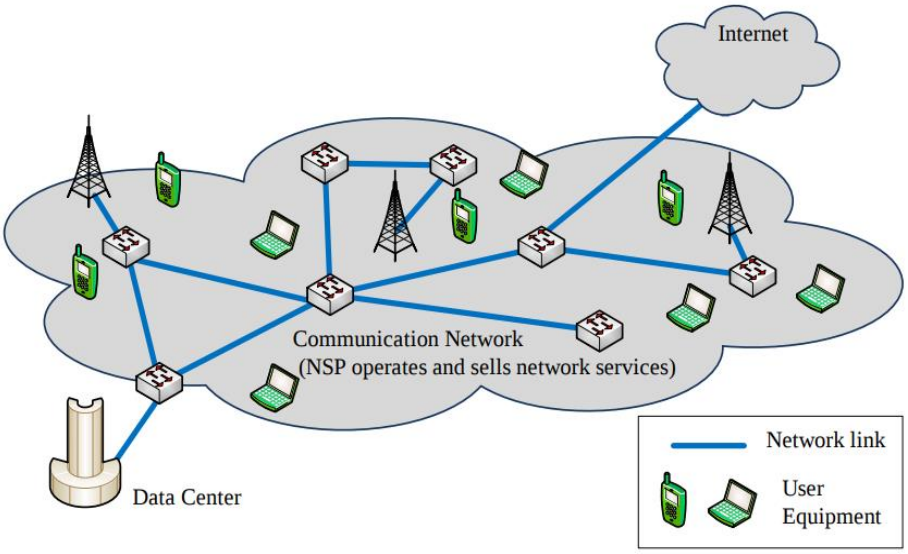


Figure 1.3.1: An ISP operating a communication network and offers a variety of network's services [28].

ISPs have a well-defined standard quality level required that must be met by the network so that a network service is provisioned to the user of the network this is known as the Quality of Services (QoS) [2].

In telecommunication networks, QoS is determined by the two main types of factors namely:

- The traditional network performance indicators such as jitter, delay, bandwidth, call setup rate, dropped call rate etc.
- The Key Performance Parameters (KPP) that are based on the quality of the service experience as perceived by the customer and not necessarily towards network performance mechanisms.

Some examples in the user oriented QoS class includes the:

- Service allocation time.
- Service Availability.
- Service maintenance time [2].

This requires that the ISPs must carefully regulate all these indicators, belonging to these classes described, to ensure that they provide an excellent QoS in their communication network components.

The installation of QoS mechanisms in communication networks needs complex configurations. This often leads to the ISPs reliance on the massive allocation of network resources to service the needs of their service consumers. From a functional viewpoint it is much easier to massively allocate the network resources to avoid service degradation experienced by the end users, compared to having to configure complex QoS mechanisms.

Network overprovisioning normally meets the need for achieving a good service quality standard on a temporary basis and it is often not sustainable in the long term for the following reasons:

- Through massively allocating the network resources, the network will finally reach the saturation point given by the expensive equipment costs and by available technologies.
- Network overprovisioning results in poor utilization of network resources.
- Network overprovisioning creates competition among services to access the limited resources in a first come first serve or best effort manner and causing QoS degradation and violation of SLA.

This means that an excellent service quality is only guaranteed if the network has plenty of resources to prevent congestion. Eventually the network will reach a saturation state when congestion emerges in the network, resulting in service quality degradation since no QoS mechanisms are deployed to ensure network services can be differentiated accordingly [3]. Due to these reasons massively allocating the network resources has proven to be not sustainable.

In order to deal with such a trend of massive allocation of the network resources instead of applying appropriate QoS techniques, the development of network technologies must focus on the simplification of the QoS allocation methods and make them available to the ISPs.

1.4 The Problem Statement and Research Questions

To motivate the ISPs to use complex QoS mechanisms in their telecommunication networks instead of massively allocating network resources, developing the latest network technologies aimed at simplifying the QoS allocation. If the QoS allocation is made simpler, it can scale up well and be applicable in huge, complex network environments.

From a comprehensive viewpoint, QoS is affected by several parameters, which shows up in the three logical network planes: - the control plane, the management plane and the data plane.

This has led to the relentless pursuit of new network technologies that will result in the simplification of the QoS provisioning through thorough consideration of several aspects that belong to all the three network planes.

SDN has two advantages which are applicable for the work in this dissertation. Firstly, the SDN can simplify the network management through programmability, since many operations can be automated with relative ease. Secondly, the centralization of the control logic in SDN permits for a more effective resource utilization compared with traditional network architecture, due to the controller's wide visibility of all the controlled network resources.

The research questions for this dissertation are formulated as follows:

- How can SDN be applied to make QoS allocation more scalable, so that it can be used in huge and complex networks?
- How is QoS allocation simplified and further enhanced by using SDN techniques?
- How can SDN provide end-to-end bandwidth guarantees between hosts?
- How to prioritize traffic correctly, thus providing guarantee to selected traffic while pushing aside excess traffic?
- How to allow excess traffic to flow in cases of excess resources?

To answer the questions listed above, the research in this dissertation aims to use the advantages of using SDN model to enhance QoS allocation for network services. The dissertation will first investigate the challenges of QoS aware service allocation in SDN.

Furthermore, the dissertation will take a comprehensive approach with respect to QoS, through exploration of the main challenges that SDN encounters at each of the logical network planes (i.e. data, control and management), with respect to scaling bandwidth management scalability and QoS allocation to huge, complex networks.

1.5 Project Constraints and Limitations

The limitations in the implementation of this project includes the hardware that is used in simulating the network.

The hardware used have the following specifications are:

- Processor: Intel Core i7-4510U CPU at 2.00GHz x 4.
- Graphics card: Intel Haswell Mobile.
- Random Access Memory: 8GB.
- Operating System: Ubuntu 16.04.

The software used includes:

- Open source programs and tools (python).
- Software emulated switches will be used and should also be open source.
- The protocol OpenFlow is to be used.

1.6 Dissertation Outline

The dissertation starts off with the introduction chapter in Software-Defined Networks and presents the problem of the bandwidth and quality of service (QoS) management in SDN environment. The project objectives are stated and the motivation for work undertaken is presented as well as the project constraints and limitations. Chapter 2, the literature survey discusses fundamental concepts in SDN including the SDN architecture, the OpenFlow communication

protocol, the concept of QoS in the SDN environment, Network Function Virtualization (NFV) and its relation to SDN, then the QoS Aware traffic classification architectures in SDN are presented and citation of the recently published work. Chapter 3 presents the SDN Provision Methods which includes the Integrated Services and Differentiated Services. The advantages of the Integrated Services and Differentiated services are presented, The Differentiated Services is chosen as the architecture of choice which provides better bandwidth and Quality of Service management capabilities in the SDN environment. The work focuses on Integrated Services and Differentiated Services architectures to investigate to choose the QoS architecture which can regulate traffic flow using open-flow protocol. In Chapter 4 and Chapter 5, DiffServ Assured Forwarding is chosen as a QoS architecture for implementation. The bandwidth management scalability in SDN is proved based on throughput analysis by considering two conditions i.e 1) Per-flow QoS operation and 2) QoS by using DiffServ operation in the SDN environment with Ryu controller. Chapter 6 presents the results and the analysis. The result shows that better performance QoS and bandwidth management is achieved using the QoS by DiffServ operation in SDN. Chapter 7 presents the conclusion and the recommendation for future work that can be pursued in the bandwidth and QoS management in SDN networks.

2. Literature Review on SDN

2.1 SDN

The latest breakthroughs in network technology development aimed at simplifying the manner in which networks are built, debugged and managed. Software Defined Networking (SDN) is an innovative model shift, which alters inherently the network devices' architecture and, hence changing the entire communication network.

The Open Networking Foundation (ONF) described the SDN as a networking product with three logical planes namely the data plane, control plane and the management plane. The data plane is responsible for the forwarding of packets from one network device to another in the network. The control plane is responsible for the implementation of the intelligence behind the forwarding the data in the network.

The architecture of the SDN networks separates the control and data plane functionalities such that the network behavior can be programmed and automated.

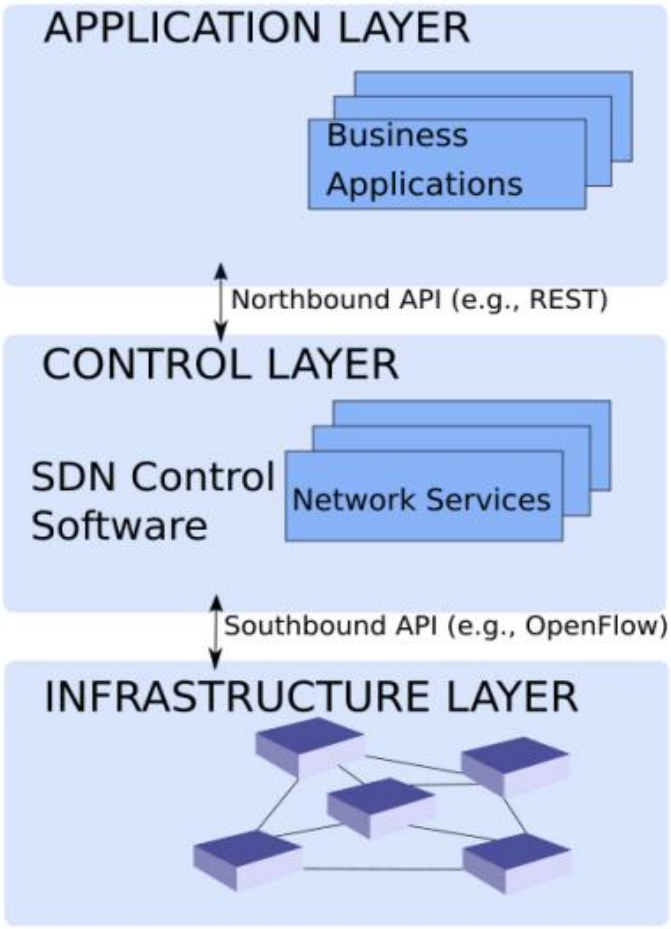


Figure 2.1.1: SDN Architecture with three network logical planes [7].

In the SDN architecture the control plane combines the Layer 2 (Data Link Layer) and Layer 3 (Network Layer) of the Open Systems Interconnection (OSI) model. The data plane is an equivalent of the Layer 1 (Physical Layer) of the OSI model.

The SDN has the following primary features:

- The decoupling of the control and data plane functionalities.
- A centralized controller that has the global view of the network resources.
- Direct network programmability is possible as the control plane functionalities are separated from the data plane functionalities.

- The decoupled control and data planes make it easier for network administrators to adjust traffic flows dynamically to cope with changing traffic conditions.
- The centralized SDN controller contains the intelligence that maintains the network topology and overview.
- SDN allows network administrators to configure, manage, enhance security measures and optimize the utilization of network resources quickly and dynamically.
- SDN is implemented on Open source tools which simplifies network design and ensure interoperability between controllers and network devices.

2.2 SDN Control Plane

The SDN architecture is based on the separation of the control plane and the data plane. The SDN architecture manages the control plane separately in the SDN controller. The SDN controller is considered as the brain of the network. The controller plays a similar role that an Operating System (OS) plays in the computer.

The main functions of the SDN controller are:

- Contains all the network intelligence logic.
- The controller is responsible for the modification of the data and for the communication between applications and network devices.
- Setting packet handling policies (e.g., security).
- Using the OpenFlow protocol the SDN controller is responsible for monitoring the network.
- Responsible for the intelligence behind the forwarding of the data.

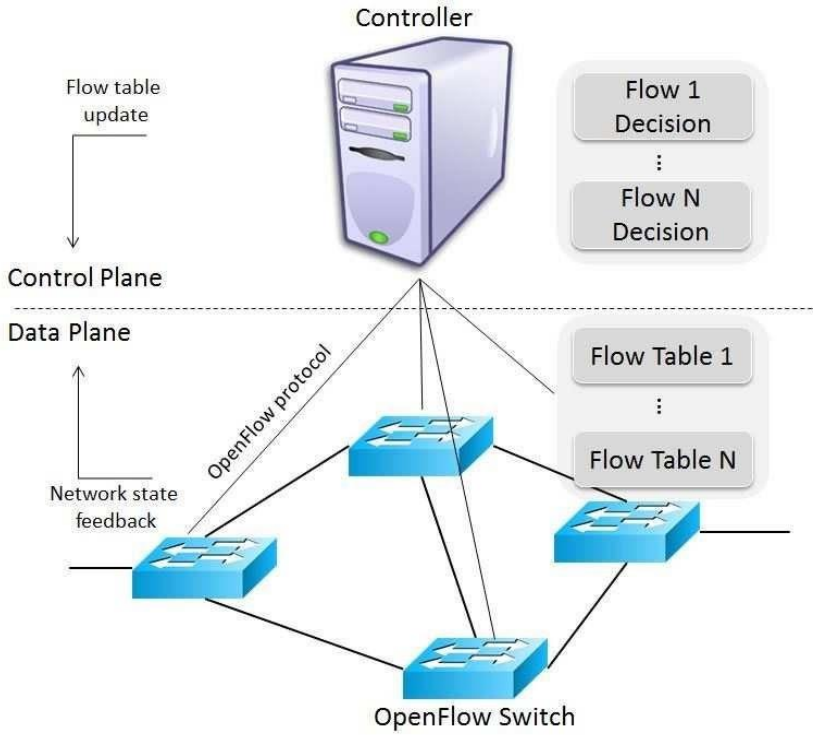


Figure 2.2.1: The OF based SDN Architecture with the Control Plane and the Data Plane [9].

The popular SDN controllers and their programming language features are given below:

- NOX SDN Controller: - based on C++ programming language with multithreading.
- POX SDN Controller: - based on Python programming language and mostly used for rapid prototyping.
- Beacon SDN Controller: - based on Java programming language with multithreading.
- Floodlight SDN Controller: - based on Java programming language with multithreading.
- MUL SDN Controller - based on the C programming language with multithreading.
- Maestro SDN Controller - based on Java programming language with multithreading.
- Ryu SDN Controller - based on Python programming language and it is good for rapid prototyping.
- OpenDaylight SDN Controller: - based on Java programming language.

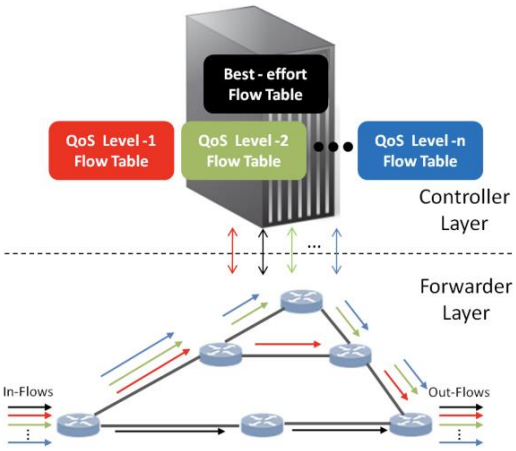


Figure 2.2.2: OpenFlow controller and forwarder interaction [31].

When an OF switch receives a new packet and whereby the switch does not have instruction and matching headers, the switch sends information about the packet to the SDN controller for processing. The SDN controller receives the packet and then decides the best way to handle it. The SDN controller can decide to drop the packet or forward it to a specific port and instruct the switch on how to handle similar packets in future.

2.3 SDN Data Plane

The main functions of the SDN Data plane are:

- All activities involving as well as resulting from data packets sent by the end user
- Forwarding of network packet from one network node to the next.
- Fragmentation of network packets and reassembling of packets at the destination host
- Replication of network packets for multicasting.
- Responsible for the forwarding of user data.
- Network packet processing operations such as traffic shaping, buffering, policing, categorization, queuing and scheduling.

2.4 SDN Management Plane

The main functions of the SDN Management plane includes:

- Providing fault tolerance mechanisms in the SDN.
- Configuration of the network nodes.
- Performing accounting activities associated with updating the SDN traffic counters.
- Ensuring that the SDN performance and utilization is efficient and optimal.
- Enforcing security measures in the SDN.
- Instantiation of new devices and communication protocols between devices.
- Network device configurations,
- Specifications on the service behaviors
- Policy specifications on how the network must behave.

The application layer consists of the SDN applications that uses the functionality offered by the SDN controller. This exchange of messages between the SDN applications in the application layer and the SDN controller happens via the Northbound Interface (NBI) which is in a form of Representational State Transfer (REST) Application Programming Interfaces (APIs) [1].

2.5 SDN Communication Protocols

SDN network layers exchange messages using the OpenFlow communication protocol. The data plane and the control plane exchange messages using the OpenFlow protocol via the southbound interface (SBI). This layer is where a controller can experience a potential bottleneck via OpenFlow.

The exchange of messages between the control plane, data plane and the management plane in SDN is made possible through the following SDN communication interfaces:

- Northbound Interface (NBI).
- Southbound Interface (SBI).
- Eastbound Interface (EBI).
- Westbound Interface (WBI).

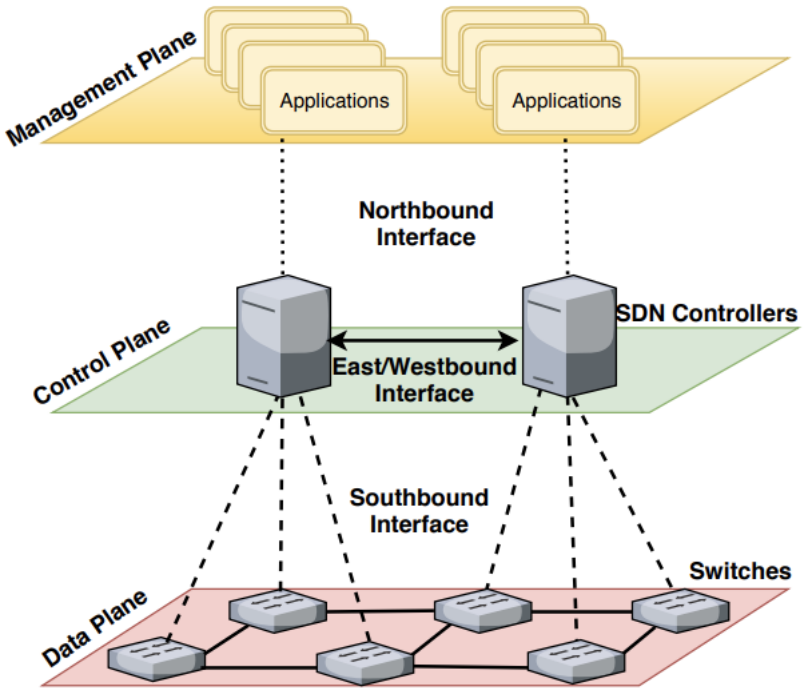


Figure 2.5.1. SDN Communication Interfaces between the Logical Network Planes. [34]

Major communication events that take place in the network plane are:

- The management/application plane, where network management operations are executed interacts with the control plane using the NBI.
- The SDN controller can program the behavior of the switches and it can also be programmed by the network administrators.
- The controller interacts with the data plane through the SBI whereby intercommunication between switches and the controller take place.

- The communication between the SDN controllers happens through the EBI and WBI in an SDN environment where multiple controllers are used.

2.6 The OpenFlow Protocol

The OF protocol is one of the vendor independent communication protocols which enables the exchange of commands between the SDN controller and the forwarding switches in the data plane [3]. The OF protocol is used for:

- Controlling the network packets forwarding behavior of the data switches.
- Maintaining and updating the switches' forwarding tables,
- Global control of the SDN and network topology discovery,
- Enabling the controller to take control of the data switches' behavior.

The architecture of the OF protocol consists of:

- OF compatible switches in the data plane.
- OF compatible SDN controller in the control plane.
- A secure channel where the interaction between the data plane and control plane occurs.

The OpenFlow Protocol supports three message types namely:

- SDN Controller -to-Switch.
- Asynchronous.
- Symmetric.

SDN Controller to Switch messages are initiated by the controller and are used for the management of the switches or for the switch state inspection. Asynchronous messages are initiated by the switch and used for updating the SDN controller of the network events and changes the switch

state. Symmetric messages are initiated by the switch or the SDN controller and sent without waiting for approval from either the switch or the SDN controller.

In OpenFlow Version 1.3, the packet headers matched using the Ethernet based TCP/IP communication protocol standards. This means that the Ethernet address, the EtherType field of the packet can serve as the basis for a match. Other packet fields which can be used as the basis for a match includes the IP version 4 addresses and transport layer ports fields of the packet [3].

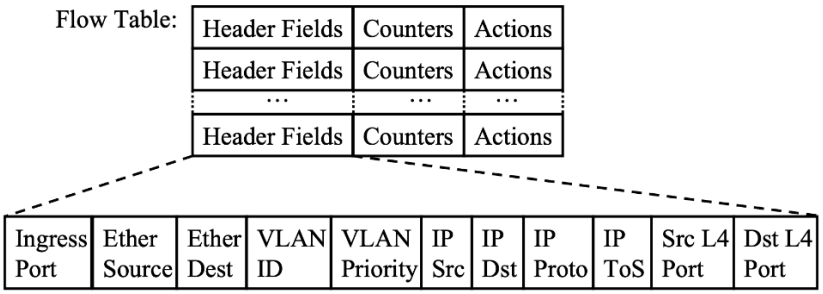


Figure 2.6.1(a) OpenFlow Packet Fields [19].

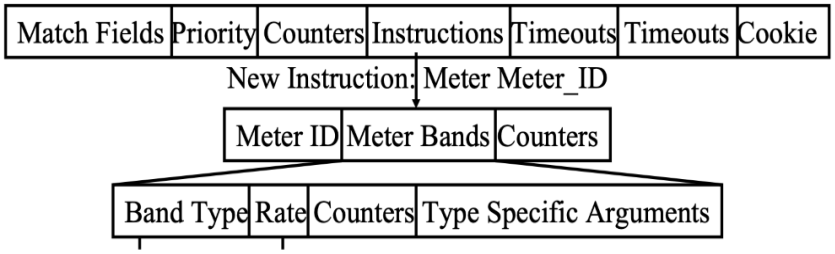


Figure 2.6.1(b) OpenFlow Packet Fields [19].

Based on the selected header fields, the granularity of the matching varies from the very coarse granularity in which the Ethernet headers are used as the basis for a match, to a much finer granularity in which the match is determined by the transport layer headers.

The mostly used actions in version 1.3 of the OpenFlow protocol includes the:

- Drop - which specifies that the packet associated with the drop flow rule entry must be dropped.

- Output - which specifies that a packet associated with the output flow rule entry must be sent out on a given port.
- Enqueue - which specifies that flow rule entry associated with the enqueue must sent the packet to the output port and the associated queue number.

Using the process of matching headers and actions, the SDN controller can:

- Regulate the forwarding of network traffic in the data plane layer through the installation of OpenFlow rule entries in the forwarding entities.
- The OpenFlow command used for the installation OF rule entries in a forwarding entity is called the FlowMod.

The flow entry rules can be installed in the SDN controller in two main approaches namely:

- Proactive approach.
- Reactive approach.

In the proactive approach the SDN controller installs the flow entry rules in the forwarding entities before such that when a new packet appear at the SDN controller, there exists an appropriate OF rule to handle and process the packet.

In the reactive approach the packet appearing at the controller has no matching OpenFlow rule in the forwarding entity. In such a case the packet is sent together with an OF Packet In message to the SDN controller. At the SDN controller the packet is then processed and the new OpenFlow rule is installed in the corresponding forwarding entity to correspond with the specified traffic flow contained in the arriving packet.

Other important OpenFlow commands includes the Statistics Request message. This message enables

- The SDN controller to poll various counters in the forwarding entity to extract different important statistics.
- The SDN controller can acquire the information on the OpenFlow entries counters and calculate the number of packets that have matched a corresponding OpenFlow entry rule.
- The SDN can then use this command to extract the information on the flow table level statistics and port level statistics.

Currently, there is a lot of forwarding device types that support OpenFlow version 1.3 and can support the latest versions of the protocol as well. A lot of effort has been invested towards the development of tools for experimentation purposes with OpenFlow based SDN. This has resulted in making it a possibility to emulate real OpenFlow devices through using software switches. A popular OpenFlow software switch in the SDN community is Open vSwitch (OVS) [5].

2.7 Communication Networks Quality of Service

Quality of Service (QoS) is defined as the capability to provide resource assurance and service differentiation in a network [6]. This is achieved using different traffic prioritization techniques or by reserving some network resources.

By assigning different priorities to different traffic flows the network can offer higher QoS mechanisms to flows with higher priority QoS requirements, for example, a video streaming application can be assigned a higher priority based on it QoS requirements (bandwidth, delay, jitter) compared to a web browsing application.

By assigning a higher priority to a specific flow, it can enable bandwidth reservation up to a defined set threshold, any application flow with a lower priority must then wait. This ensures that bandwidth guarantee to chosen hosts with higher priority QoS requirements. The application of priority-based approach can also be used to avoid network congestion.

By assigning a lower priority to chosen traffic in the cases of network congestion, the network can then decide to drop low priority traffic flows to avoid network congestion and make room for those traffic flows with high priority [7].

The QoS in telecommunication networks relies on the ISP based parameters and the customer-based parameters. ISP-based parameters refer to the underlying low-level network mechanisms. The ISP based parameters are important for:

- They determine how the network the quality indication of the service being offered to the end users.
- They are referred to as Network Performance parameters which include parameters such as delay, bandwidth, and jitter.
- The NPP are the basis upon which the QoS can be quantified by the ISP.

The customer-based parameters are important for:

- They gauge the user level of satisfaction in using a service.
- They are called the Non-Network Performance parameters (NNPP).

Examples of customer-based parameters include:

- Service availability - depends on how easy it is to access the service at any point in time.
- Service maintenance time.
- Ease of using the service.

Most of the telecommunication networks are designed to include regulatory mechanisms for all these factors to ensure that the QoS level standard is met for different services.

The regulation of the NP parameters and the NNP parameters towards achieving QoS in telecommunication networks suggests that mechanisms must be implemented at all the network logical planes. In the data plane, packets processing must be done according to their specified level of quality.

Control plane mechanisms must function such that an efficient usage of network resources is maintained, and with as minimal disruptions on the availability of services as possible. Management plane operations must be fast in their nature to reduce the service allocation time. Ideally, the service allocation should be automated to eliminate human intervention in order to increase the adoption of QoS mechanisms in telecommunication networks.

The research conducted in this dissertation uses an OVS as an OpenFlow forwarding device for the experiments conducted in the designed chapter. This chapter presented an introduction into OF based SDN. The following chapters will address the challenge of offering a QoS aware network service in SDN. The other details of the OpenFlow protocol and SDN behavior will be systematically introduced through the dissertation and the context within which they are applicable towards solving the QoS challenges in the SDN.

2.8 Network Function Virtualization and SDN

Software-defined networking and Network Functions Virtualization (NFV) are enabling network architects to design, implement, and manage network services efficiently. Network architects and administrators use software for configuration and management of network functions through a centralized point [35].Network Functions Virtualization (NFV) decouples network functions from proprietary hardware appliances and delivers an equivalent network functionality without the necessity for specialized hardware.

2.8.1 NFV and SDN Similarities

NFV and SDN are interdependent in multiple ways, and when deployed together can achieve flexible, agile network infrastructures. NFV provides the basic networking functions and SDN provides higher level management responsibility to organize the overall network operations.

- SDN deployment – runs on virtual machines, hypervisors, network controllers, load balancers, and gateways are deployed and configured to provide the needed network infrastructure controls and regulation policies.
- NFV deployment – a wide range of virtualized network functions such as routers, firewalls are deployed as software on top of a virtualized infrastructure.
- SDN Management – centralized control console to monitor throughput, routing and policy definitions.
- NFV Management – virtual network functions are centrally managed and monitored regardless of their location across the network.

- SDN Costs - primary costs savings come from the reduction of operational expenditure through the automation of network configuration and modifications. Personnel costs account for much of overall spend. A small reduction in operational costs can result in a significant cost benefit.
- NFV Costs – running on high performance servers in datacenters, virtual network functions eliminate the necessity of procuring specialized network hardware for each individual network function. This allows for less space, power, cooling and equipment to be deployed.
- SDN Flexibility – programmable interfaces enable provisioning of new network devices, reconfiguration of existing devices through scripting and management consoles.
- NFV Flexibility – quickly deploy and commission functions to support proof of concept trials. Locate functions at the network edge, close to data, applications and users to optimize network security and performance.

2.8.2 NFV and SDN Differences

SDN and NFV have much in common, based on the concept of virtualization that drives the development and deployment of their capabilities. The main differences between SDN and NFV concern the overall focus on network management responsibilities and the standards that guide the architectural and functional development.

- SDN Scope – defines the big picture aspects of the entire network - the type of infrastructure, services and applications available. Determines network policies that guide

delivery and use of network resources. Hypervisor orchestrates and controls lower-level network functions.

- NFV Scope – deliver a wide range of specific functionalities that must be performed at all levels and stages of the network, at the periphery, boundary and core under the control of a hypervisor.
- SDN Standards – Open Networking Foundation (ONF) is responsible for developing open standards, vendor-neutral standards, for the communications interface defined between the control and forwarding planes of an SDN architecture.
- NFV Standards - European Telecommunications Standards Institute (ETSI) defines and maintains globally applicable standards for information and technologies regarding NFV.

2.9 QoS Aware Traffic Classification Architectures in SDN

To achieve QoS aware traffic classification, various techniques have been presented in prior research. Prior techniques can be classified into three categories. The first category is based on deep packet inspection (DPI) mechanism [36] [37]. The DPI techniques can accurately detect the data packet payload; thus, the controller can identify the traffic flow and make QoS classification. DPI is limited by the encryption of packet payloads, private protocols, peer-to-peer port encryption and other restrictions. It is very difficult to identify all of the data flows and identifying the underlying protocols requires a lot of reverse engineering. Network applications emerge rapidly and a lot of these applications provide homogeneity services in practice, thus the QoS requirements of those applications are similar. It is inefficient to identify each specific application with DPI and moreover, maintaining a database

that contains all web applications is impractical. In the SDN network the traffic classification must be real time and low cost. DPI technology consumes a large amount of computing resources and makes the network to incur a significant amount of delay which reduces the network reaction speed.

To achieve QoS aware traffic classification, various techniques have been presented in prior research. Prior techniques can be classified into three categories. The first category is based on deep packet inspection (DPI) mechanism [36] [37]. The DPI techniques can accurately detect the data packet payload; thus, the controller can identify the traffic flow and make QoS classification. DPI is limited by the encryption of packet payloads, private protocols, peer-to-peer port encryption and other restrictions. It is very difficult to identify all of the data flows and identifying the underlying protocols requires a lot of reverse engineering. Network applications emerge rapidly and a lot of these applications provide homogeneity services in practice, thus the QoS requirements of those applications are similar. It is inefficient to identify each specific application with DPI and moreover, maintaining a database that contains all web applications is impractical. In the SDN network the traffic classification must be real time and low cost. DPI technology consumes a large amount of computing resources and makes the network to incur a significant amount of delay which reduces the network reaction speed.

The second category is based on Machine Learning (ML) mechanism. The emergence of ML provides a new method of traffic classification. Compared with the DPI mechanism, ML flow classification mechanism needs to extract the characteristics of the flow (size of previous N packets, source IP address, protocols or flow arrival interval) rather than checking payloads of the packets so that encrypted flows can be classified. ML mechanism is less computationally complex,

[38] [39] [40] [41] use supervised learning training mechanism to train the classifier while [42] use neural networks for feature extraction and training the classifier. [43] [44] proposed an unsupervised learning framework for training the classifier. Supervised learning requires that all training data are correctly labelled as known applications which is not realistic in the real-world environment. Conversely, data in unsupervised learning don't have specific application label. It is impractical to achieve low complexity classification based on unsupervised learning.

The third category coordinate multiple classification mechanism to achieve traffic classification. It combines the advantages of several mechanisms to classify traffic. For example, [45] combined the DPI and port number classification mechanism. Most of the flow classification techniques are limited to the low recognition accuracy and prior techniques are incapable to deal with the rapid emergence of new applications in the network [46].

3. SDN QoS Provisioning Methods

SDN architecture offer similar services as conventional telecommunications network architectures but using the OpenFlow protocol. This chapter provides an insight into network services provisioning methods used in an OpenFlow based SDN. The focus is on the methods used to implement QoS mechanisms for SDN services. The OpenFlow protocol allows for regulating the traffic in the network, but it is often not enough for offering complex QoS implementations that require more advanced network resources configurations. The challenges related to the simplification of QoS configuration mechanisms in SDN are given with the purpose of offering SDN flows with good QoS.

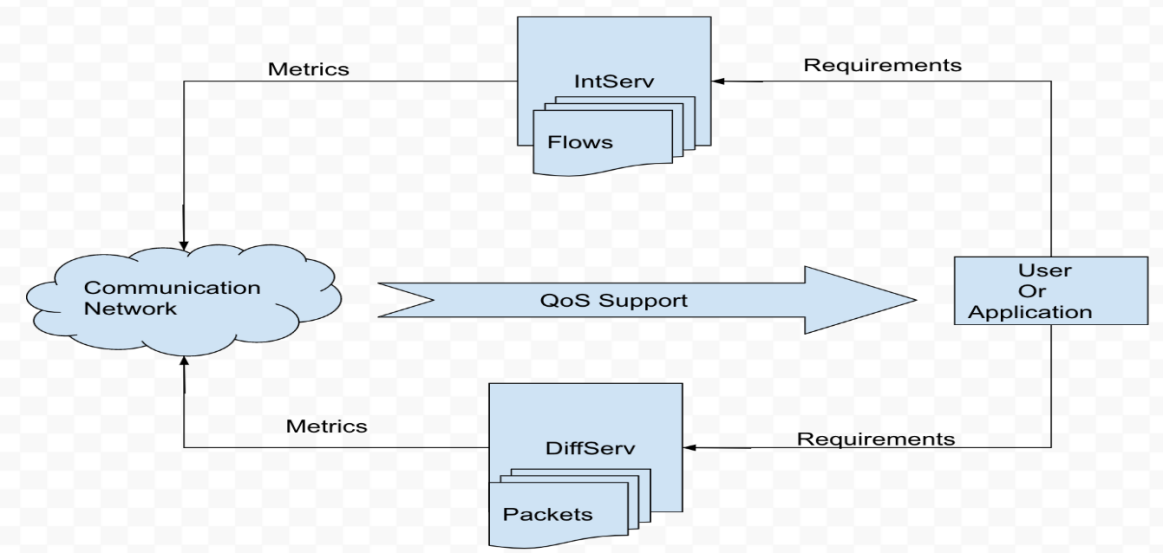


Figure 3.1: SDN QoS Provisioning Methods

The QoS architectures can be classified into:

- Integrated Services which provide hard QoS guarantees using resource reservation techniques (bandwidth, buffer). [8]
- Differentiated Services which provide hard QoS guarantees using resource reservation techniques (bandwidth, buffer). [9]

The Integrated Services and Differentiated Services architectures are examined and evaluated for the purpose of choosing the QoS architecture for this project.

3.1 Integrated Services

Integrated Services (IntServ) is an architecture that specifies the elements to guarantee QoS for individual flows. In IntServ, a flow is defined as a unidirectional data stream between two applications and is uniquely identified by source and destination address pair, port numbers and the transport protocol. Integrated Services (IntServ) uses resource reservation protocol to ensure QoS requirements are met [8]. Each router in the IntServ network domain follows and adheres to the same resource reservation policy.

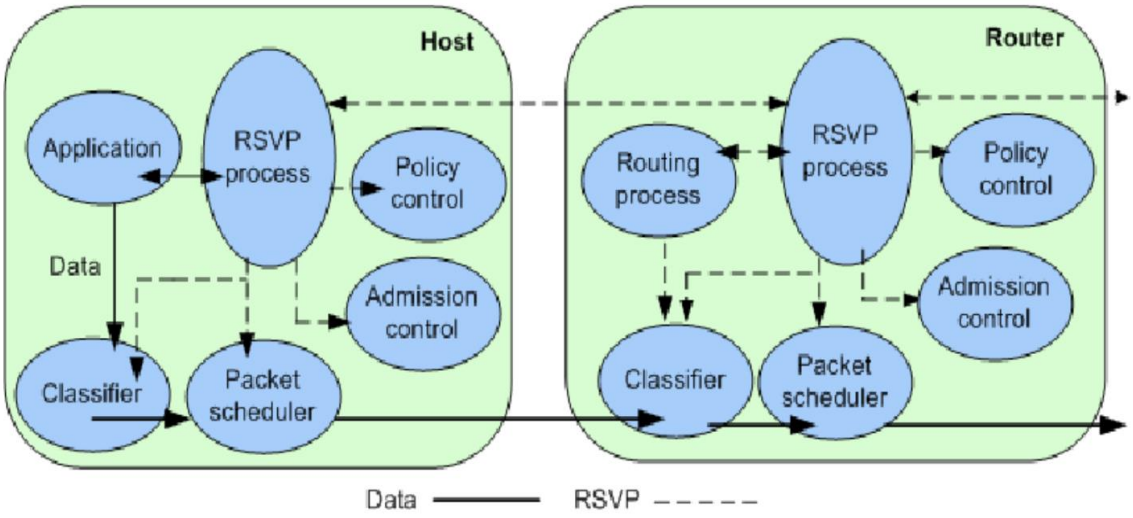


Figure 3.1.1: IntServ's QoS Architecture in Host and Router in SDN context [15].

IntServ in the following way:

- Each network application that has QoS requirements to be satisfied must make a reservation request in all routers in the path between the application host and the destination host.

- Each router receives the resource reservation request and must respond if they satisfy the QoS requirements for the requesting application.
- Each router between the sender and the receiver must accept the request for the guarantee to be realized.

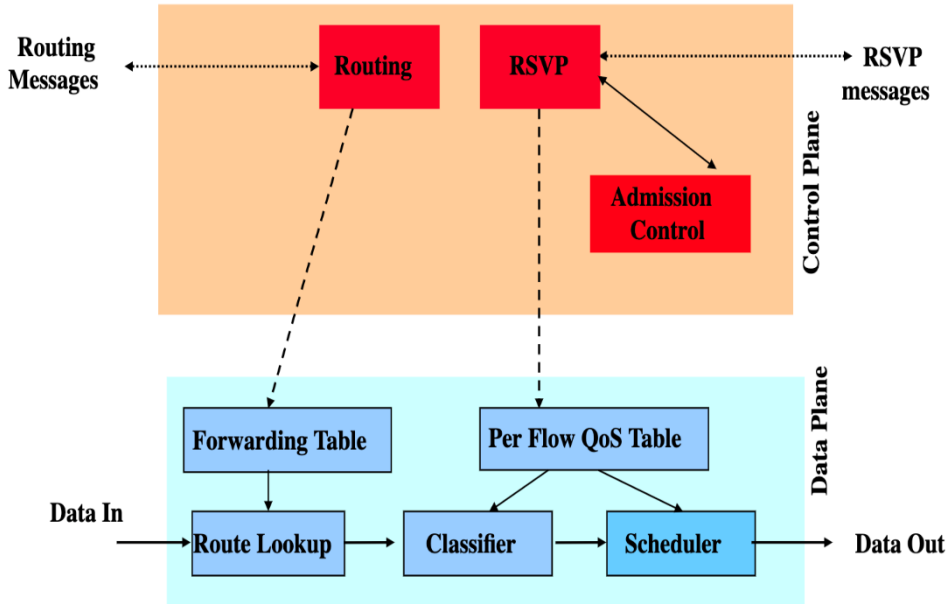


Figure 3.1.2: IntServ's QoS Node Architecture [16].

In Intserv, there exists three classes of services which are based on the application's delay requirements. The services are assigned priorities from highest priority which is given to applications with stringent delay requirements to lowest priority which is given to applications with delay requirements which are not stringent.

IntServ is consists of two functions namely.

- Flow Spec -which is used by applications to specify the required QoS reservation for the routers.

- Resource reservation protocol (RSVP) which is used to communicate the requests and acknowledgements between routers and requesting applications.

FlowSpec is divided into two parts which are:

- Traffic Specification (TSPEC) - used to provide details about the nature of traffic from the application. Details such as frame sizes and packet sizes are furnished so that router can reserve network resources accordingly.
- Request Specification (RSPEC) -is used to specify the QoS requirements of the flows.

The three service classes in IntServ are:

- Guaranteed service [10] – which ensures that firm bounds on end-to-end datagram queuing delays are satisfied and offers hard QoS guarantees (in terms of delay and bandwidth requirements) for each flow.
- Controlled Load service [11] – which ensures that an approximate QoS guarantees are satisfied in a moderately loaded network. The delay service agreement is measured in a statistical way and is not violated under unloaded network conditions.
- Best-effort service – which provides the service like that of the Internet in a first come first serve manner with no priorities assigned to different network services' traffic.

The guaranteed service and controlled load traffic classes are based on measured service requirements, and they require signaling to reserve network resources such as link bandwidth, path delay estimation and buffer size. Resource Reservation Protocol (RSVP) [12] is used as the signaling protocol in IntServ.

RSVP is made up of types of messages used for communication:

1. PATH – This message is used by the source host that has traffic which requires QoS requirements to be met when sending to a receiver.

- The PATH message is used for identification of data paths and sends the TSPEC of the sender.
- resv: Reservation Messages. The recipient of a PATH message can reply with a resv message to initiate a flow.
- The resv message consists of the FlowSpec which is sent through the data path of the PATH message as a reply to the sender.
- From the resv message, the routers then reserve the appropriate amount of resources for the flow between a sender and receiver.

IntServ has an internal Soft State method that ensures that the reservations are cancelled if nothing is transmitted from the sender after a set time. This helps dismiss reservations in the event of link failure like crashes.

RSVP is used:

- For bandwidth reservation purposes for the applications requesting QoS for their unicast of multicast data flows.
- By a host to request specific QoS requirements from the network.
- By routers to exchange and share the QoS requirements of the host with the routers in the same path.
- For establishing the end to end connection before the data transmission starts between source and destination hosts in the network.

RSVP functions in the SDN includes:

- It interoperates with any unicast and multicast routing protocols.
- Routing protocols decides where packets get forwarded and RSVP deals with the QoS management of the packets that are forwarded in accordance with a chosen routing protocol.

It is essential that the routing protocols must be QoS-aware, to ensure that the calculated routes satisfy the QoS requirements. In an IntServ QoS architecture routers must execute the following traffic control functions to meet specific QoS requirements for each flow:

- Admission control function – decision on whether to admit or reject an application's request depending on the availability of network resources. Admission control needs that the router understands the QoS requirements that are currently being requested on its network resources, so that it can forecast the worst-case bounds on each service provision.
- Packet scheduler – implements the management of forwarding packet streams using a set of queues and buffers and other mechanisms such as counters.
- Packet classifier – implements the traffic control, incoming packet mapping into a traffic class.

The advantages of IntServ includes:

- Guarantees service that satisfies QoS requirements with firm bounds on delay.
- Guarantees the end-to-end QoS requirements on per-flow basis is achieved.

The IntServ architecture has the following disadvantages and drawbacks:

- Scalability - IntServ operates well on relatively small-scale networks

- It is hard to control and keep track of all network resource reservations and end-to-end signaling in a large-scale network.
- IntServ implementation requires fundamental changes in the core network, because all routers along the chosen traffic path must support it.

3.2 Differentiated Services

Differentiated Services (DiffServ) is an architecture that outline the mechanisms for classifying, management of data traffic and offering QoS for aggregated traffic classes. DiffServ is a network architecture that uses Differentiated Services Code Point (DSCP) values for classification of packets that goes through its domain. The DiffServ domain refers to a set of routers that use the same Diffserv policies [16].

A DiffServ domain consists of:

- A set of connected DiffServ nodes that use the same service policy and per hop behaviours (PHBs).
- All routers in the DiffServ domain are configured to differentiate traffic based on the assigned priority given to the incoming packets.

The DiffServ architecture is based on the principle of traffic classification. The principle is that every packet is placed into a chosen number of traffic classes and traffic classes are prioritized differently. The routers in the DiffServ domain uses the DSCP value in the IP-header to decide on how to differentiate the traffic.

Compared to IntServ's fine-grained and flow-based routing mechanism, DiffServ is based on the coarse-grained and class-based mechanism for traffic management in the network.

In DiffServ QoS architecture, network packets are classified and marked to receive a per-hop forwarding behavior on all the routers along their path.

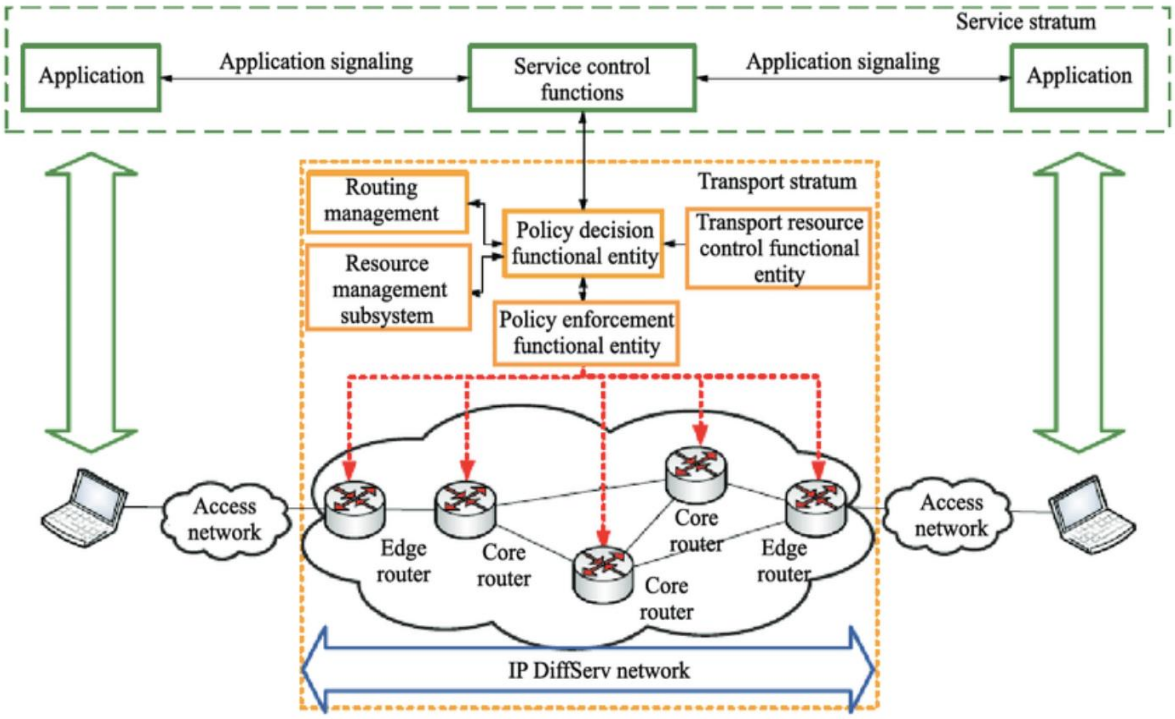


Figure 3.2.1: Architecture of the DiffServ QoS in SDN [17].

The main advantages of DiffServ QoS architecture over IntServ QoS architecture is the scalability which is achieved by implementing operations which includes:

- Packet classification.
- Packet marking.
- Traffic shaping and policing at network hosts.
- It does not need fundamental changes in the core network.
- Routers in the network are only configured to differentiate traffic based on its class.
- Each traffic class is managed based on its assigned priority preferences.

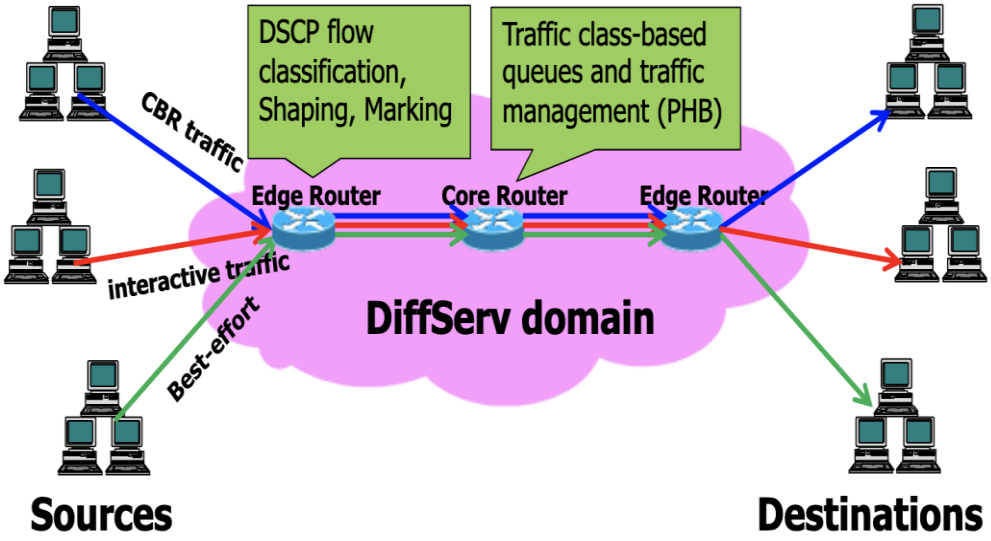


Figure 3.2.2: DiffServ's QoS Architecture [18].

A DiffServ QoS based network must implement:

- Network packet classification and packet marking on the egress and ingress routers.
- Per-hop behaviors (PHBs) such as priority queueing and traffic shaping at all the routers in the network.

Per-hop behaviors are essentially for:

- They offer soft guarantees with statistical bounds on end to-end delay.
- PHB is used to define the packet-forwarding attributes that is associated with each of the classes.
- PHB is used to apply the policies and priorities to a packet of selected class when performing a hop.
- It does not fully guarantee end-to-end QoS, it offers the requested QoS with high probability. This allows adjustable class prioritization assignment and enables service differentiation.

- The PHB behavior is determined by both the DSCP value and Explicit Congestion Notification (ECN) bits.

The mostly used per hop behaviors used in modern networks are based on priority-based include:

- Expedited Forwarding (EF) PHB [13] which is devoted to low loss, low latency traffic such as video streaming, voice over IP. It is implemented using priority assignment queuing along with rate limiting on a traffic class.
- Assured Forwarding (AF) PHB [14] which offers assurance of packet delivery under specified conditions.
- Class Selector PHBs [15] which is responsible for maintenance of backward compatibility with the IP Precedence field for network nodes implementing IP-based precedence-based classification and forwarding.
- Default PHB [15] which refers to the best-effort traffic.

The main advantages of DiffServ includes:

- DiffServ is scalable in guaranteeing that QoS requirements are satisfied by executing policy enforcement and classification functions at the boundaries of DiffServ domains.
- It requires minimal changes in the core network.
- DiffServ does not need complex setup, configurations, reservation and time-consuming end to-end negotiation for each flow.
- DiffServ allows adaptable class service definitions and differentiated pricing of telecommunication services.

The disadvantages of DiffServ architecture include:

- The DiffServ-aware routers apply per-hop behaviors (PHBs) to traffic service classes, it is hard to forecast the end-to-end behaviour of the network.
- The problem is complicated if there are multiple DiffServ domains.
- DiffServ does not offer hard guarantees for QoS requirements.
- It ensures statistical bounds on QoS parameters such delay and throughput are met.

3.3 IntServ and DiffServ QoS Architectures

Both IntServ and DiffServ QoS architectures allows for per-flow QoS and thus, per-flow bandwidth guarantee, which forms part of the project objectives.

The main differences in which they achieve the QoS requirements in the network include:

- DiffServ achieves QoS requirements through using DSCP to assign different priorities to traffic flows.
- IntServ achieves QoS requirements by making each router in the flow path to reserve resources for each specific flow.
- The packet handling in IntServ is also done on per flow basis.

IntServ is not well suited for use in a large-scale network setup because:

- Each router stores the state information and reservation from all flows that in transit.
- Each router must store a lot of reservations and it will become hard to keep track of everything.

- The reservations are unique from one router to another since they can have unique flows running through them, making it even more difficult to keep track of every reservation.
- Flows are not consistent sometimes resulting in IntServ dropping reservations if the period of idleness occurs to make space for new reservations.
- In cases of inconsistent traffic, the applications must request reservations every time they want to send something, making it unnecessary activity.

DiffServ has the following advantages over IntServ:

- It is class based; the routers only need to know the operations to do when a specific traffic class arrives. Every router has the same class configuration.
- It is relatively easier to keep track of and to make configurations.
- It makes the network more scalable, since the class definitions don't change when more flows are created.

	Best Effort	DiffServ	IntServ
Service	• Connectivity. • No isolation. • No guarantees.	• Per-aggregation isolation. • Per-aggregation guarantee	• Per flow isolation. • Per flow guaranteed.
Service Scope	• End –to-end.	• Domain.	• End-to-end.
Complexity	• No setup	• Long term set up	Per flow set up
Scalability	• Highly scalable • Nodes maintain only routing state.	• Scalable. • Edge routers maintain per aggregate state. • Core routers maintain per class state.	• Not Scalable. • Each router maintains per flow state.

Table 1: Summary of the comparisons between the QoS Architectures in SDN.

3.4 Differentiated Services Code Point (DSCP)

DSCP is used to assign priority levels in different traffic flows. The important properties of how DSCP works are:

- Six bits are used to classify different priority levels of IP-packets.
- The 6-bits are contained within an 8-bit field called Differentiated services field (DS field).
- The other 2-bits in the DS field is called Explicit Congestion Notification (ECN).
- ECN alters how network signals congestion.
- With ECN, networks can signal network congestion by marking the 2 ECN bits in the DS field.
- Networks signal network congestion results into dropping packets, and ECN allows notification through the network without dropping any packets [17] [18].

The DS field consists of the DSCP bits and the ECN bits. The DS fields is found in the IP header of a packet. IPv4 Type of Service (TOS) field can be used for assigning priorities to different traffic types. The DS field is the equivalent of the TOS field [19].

3.4.1 IP Precedence Values

The Precedence values are consisting of a 3-bit field which is used to determine the priority of the packet. The field ranges from 0 to 7, 0 is considered the lowest priority and 7 considered the highest. The eight values are given in Table 2 below:

Binary Value	Decimal Equivalent	Service Class Description
000	0	Best Effort
001	1	Priority
010	2	Immediate
011	3	Flash
100	4	Flash Override
101	5	Critical
110	6	Internet
111	7	Network

Table 2: Priority Precedence Values

DSCP values are compared to the equivalent IP Precedence Values, as they DSCP values and DS they define the type of service with which the packet will be handled. DSCP value contains 6-bits, which translates to it 64 possible values and consequently 64 traffic classes [20]. The PHBs are not encoded, allowing for flexibility in defining specific behaviors for different traffic classes.

Networks uses three commonly defined PHBs namely:

- Default Forwarding (DF).
- Expedited Forwarding (EF).
- Assured Forwarding (AF).

Default Forwarding (DF)	Expedited Forwarding (EF)	Assured Forwarding (AF)
• Used for traffic that does not meet any of the requirements of the defined classes in the network is placed. • Implements no prioritization when handling traffic from different applications. • Implements first come first serve when assigning network resources to applications. • Applications compete for QoS resources resulting into network congestion and network delays • The traffic is placed as best-effort traffic [29].	• Used for traffic with QoS requirements that are low latency and low packet loss. • Assigns priorities to different traffic classes. • Restricted use as the applications using EF are very sensitive to latency and packet drops. • Assigning too much traffic on the same priority level results in overloads within the priority level and resulting into delays. • Used in applications like VoIP, video streaming	• Used to ensure the delivery of the packets, while the traffic is kept within a specified set of constraints. • Traffic that exceeds the given QoS constraints risks being dropped in cases of network congestion. • Services being dropped depends on the assigned DSCP value. • Suitable for use when implementing bandwidth guarantees of services. • Ensures packet delivery within a given set of bandwidth constraints.

Table 3. Characteristics of the common PHBs DF, EF and AF

Differentiated Services and Integrated Services architectures were examined, evaluated, compared for the purpose of choosing the QoS architecture for this project.From having weighed comparisons between IntServ and DiffServ QoS architectures, the class based QoS is considered the best approach based on its merits and thus, DiffServ AF is the chosen QoS architecture for implementation in this project.

4. DiffServ SDN QoS Architecture

This dissertation report uses the network laboratory facilities to implement and emulate the network topology for the purpose of conducting network topology performance metrics experiments, to prove scalability achieved using flow aggregation and bandwidth management in SDN environment.

AF consists of four different classes and each class has three different drop rates, resulting into 12 different DSCP values that can be used to tune DiffServ domain. The four classes are assigned different priority levels. The classes only with higher priority levels are prioritized in terms of network resources reservation in the cases of network congestion as the traffic is divided between different classes. This is achieved using the weighted fair queuing (WFQ) algorithm.

4.1 Weighted Fair Queuing Algorithm

WFQ is an algorithm that separates traffic into weighted classes. Every packet is assigned a traffic class and placed into a class specific queue which receives a rate that depends on its assigned weight compared to the total sum of weights between all classes. WFQ is used to deliver equal rate divided to all flows [21].

The rate that each weight class received is computed in the following way:

$$R(i) = R \times \frac{w_i}{\sum_{k=1}^{n} w_k} \quad (1)$$

Where:

- R represents the maximum link rate
- W refers to the weight of a given flow.
- The weight is divided by the sum of all weights to give a fraction of the rate for the flow to receive.
- The weight of each individual flow is configured such that higher priority classes are assigned a higher weight to ensure priority treatment between the classes.
- n represents the total number of flows.

The drop rate specifies the probability that the network packets are dropped in cases of network congestion if the traffic has been assigned the same priority class level.

The drop rates that exist are:

- Low.
- Medium.
- High.

If the network experiences congestion, packets with higher drop probability are dropped. To avoid issues with Tail drop, Random early detection (RED) is devised when dropping the network packets.

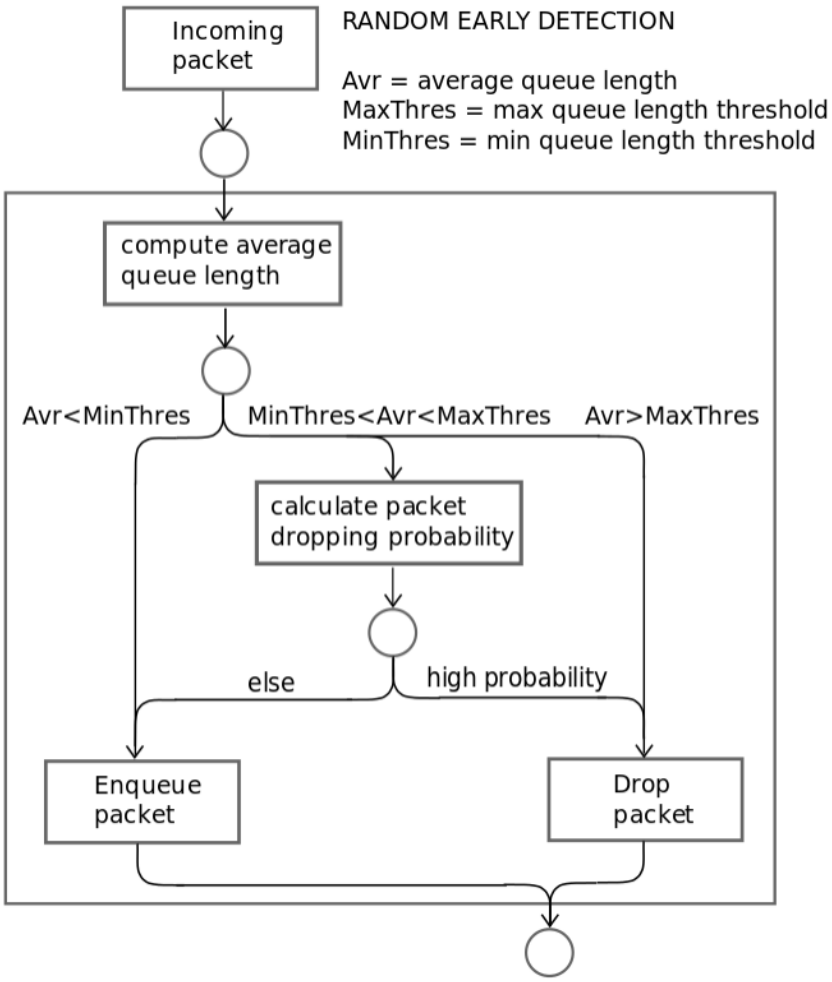


Figure 4.1.1: Random Early Detection Algorithm Flow Chart [19].

4.2 Random Early Detection and Tail Drop

Tail drop is one of the basic queuing management algorithms. It works in the following way:

- Allow the queue fill up.
- Drop each incoming packet until the queue has free space to accommodate new packets [22].

The algorithm is not efficient, since it does not assign any special treatment to any priority class levels, as it drops every incoming packet. It is unfairly biased towards packets of bursts nature. The bursts quickly fill up the queue and occupying the space, resulting in the dropping of other packets.

Tail dropping network packets can create TCP Global Synchronization (TGS). This happens when numerous TCP connections have their packets dropped as a result of tail drop. In that case each of the connections must reduce their transmission rate and battle with the congestion at the same rate, which results into poor utilization of network resources due to reduced transmission rates from the TCP hosts.

The results of reduced transmission rate can cause:

- TCP hosts to start ramping up their transmission rate at the same time, which then leads to congestion.
- They will then reduce the rate in order to battle the congestion, and now a cycle of unused network resources overflow will follow [23].

Tail drop has no QoS mechanisms implementation and cannot be used in conjunction with PHB. RED has the following benefits over Tail drop:

- RED provides some forms of QoS mechanisms applied to the queuing system.
- Helps with avoidance of TGS.

RED operates differently than Tail drop and this is shown here:

- RED is based on the computation of the average queue size with a minimum and maximum threshold for that average value.
- For each incoming packet it computes the queue average size and compares it to the maximum threshold and the minimum threshold.
- If the mean value lies between the minimum and maximum thresholds, packets are then marked with a drop probability
- Marking packets with drop probability increases with the amount of bandwidth the connection has.
- The more packets marked inside the queue for a given connection, the higher is the probability of it getting dropped.
- The packets are then dropped with the calculated probability.
- In a case whereby the mean is lower or higher than the thresholds, no probability is applied.
- If the average is lower than minimum, nothing happens.
- If the average is larger than maximum, all packets are dropped, which is like tail drop.

The RED algorithm executes on every incoming packet is summarized in the pseudocode below:

```
1
2     Average queue size calculation: avg_size
3     if( max_threshhold ⩾ avg_size > min_threshold)
4     {
5          Calculate drop probability p
6          Drop with probability of p
7     }
8     else if( avg_size > max_threshold )
9          Drop
10
```

Figure 4.2.1: RED Algorithm.

The Drop probability ranges from 0 to 1 and is computed as:

$$P_1 = \frac{avgSize - minThreshold}{maxThreshold - minThreshold} \quad (2)$$

$$p = \frac{P_1}{1 - count \times P_1} \quad (3)$$

Where:

- count is the number of packets received counting from last dropped packet.
- p represents the drop probability .
- P1 is a configurable value in the range between 0 and 1.
- minThreshold – specifies the average queue size below which no packets will be marked.
- maxThreshold – specifies the average queue size above which all packets will be marked.
- The optimal values for minThreshold and maxThreshold depends on the desired average queue.
- For bursty traffic the minThreshold value should be large to allow the link utilization to be maintained at an acceptable high level.
- The optimal value for maxThreshold depends on the maximum average delay that can be allowed by the router.

4.3 Class Selection

The DS field serves as the equivalent version of TOS field and the DiffServ still requires backwards compatibility with networks that are still using the old TOS fields. The backwards compatibility is important so that networks with DiffServ can be integrated harmoniously with

networks that are still using TOS. The Class selector (CS) PHB is defined to ensure that backwards compatibility is possible.

TOS field uses 3-bits and the CS uses the first 3 bits in the 6-bit DSCP field to match. The CS values ranges from 0 to 7 to match the assigned TOS priority levels. The three last bits in the DSCP field are always set to for DiffServ to identify that the DSCP value it reads are from a TOS network. When the last 3 bits in the 6-bit fields are 000 the CS PHB is applied. When the last 3 bits are set to something different from 000 then either DF, EF or AF are applied.

DSCP Binary Value	DSCP Decimal Equivalent Value	Assigned Equivalent TOS	Priority Level Assignment
000 000	0	Best Effort	0
001 000	8	Priority	1
010 000	16	Immediate	2
011 000	24	Flash	3
100 000	32	Flash Override	4
101 000	40	Critical	5
110 000	48	Internet	6
111 000	56	Network	7

Table 4: Traffic Class selection Values and Priority Level Assignment.

The table 5 below depicts the DSCP values are commonly used in networks [28].

DSCP Binary Value	DSCP Decimal Equivalent	PHB	Drop Rate	Precedence value	Priority Assignment
101 110	46	EF	Not Applicable	101	Critical
000 000	0	Best Effort	Not Applicable	000	Routine/ Default
001 010	10	AF11	Low	001	Priority
001 100	12	AF12	Medium	001	Priority
001 110	14	AF13	High	001	Priority
010 010	18	AF21	Low	010	Immediate
010 100	20	AF22	Medium	010	Immediate
010 110	22	AF23	High	010	Immediate
011 010	26	AF31	Low	011	Flash
011 100	28	AF32	Medium	011	Flash
011 110	30	AF33	High	011	Flash
100 010	34	AF41	Low	100	Flash Override
100 100	36	AF42	Medium	100	Flash Override
100 110	38	AF43	High	100	Flash Override

Table 5: Commonly Used DSCP values in Networks.

4.3.1 DSCP Remarking

The knowledge of how to enable QoS in SDN has been gathered and the implementation of it is undertaken here. OF offers support in marking packets with different DSCP values of choice. In the DiffServ QoS architecture using AF QoS is implemented using DSCP Metering which is discussed and implemented below.

4.3.2 DSCP Metering

OF version 1.3 introduced the concept of metering, which enabled SDN operators to pass flows into specified meters that perform actions upon those flows [24]. These actions allow for the implementation of QoS mechanisms in SDN. Actions such as DSCP remarking based on flow rate is enabled with metering in OF version 1.3.

The concept of metering contains two parts:

- Meter Tables.
- Meter Bands.

4.3.3 Meter Table

A meter table is made up of numerous meter entries. Each entry in the meter table is associated with a unique flow. This allows the capability to impose different operations to each flow. It also allows the capability to regulate each unique flow, QoS mechanisms can be implemented. Using meters, the DSCP values can be altered to suit the flow rate, resulting in that a DiffServ domain can be created.

The meter table works in the following way:

- Each flow that has a meter must pass and using the meter and meter bands before it gets forwarded.
- The meter measures the rate of each flow that passes through.
- This allows for options to impose operations based on rates with the help of meter bands.
- The meter entries are attached to each flow so that it can distinguish flows from similar ports. It is not limited by the number of ports to perform QoS operations.
- The operations are based on the number of flows.
- It does not group up every flow that belongs to the same port.

A flow does not need to be attached to a meter entry. It is generally left with the developer to select which flows, or type of flows that meters should be attached to entries and passed through the meters. A flow can also go through numerous meters. Each flow cannot be attached to numerous meters simultaneously, but it can be used in succession. This is achieved by using different meter entries in different flow tables.

A meter entry is made up of three components namely:

1. Meter Identifier - The Meter ID is a 32-bit unsigned identifier which is used by flows for identification of meter entry class.
2. Counter - A normal counter that keeps track of the number of packets that has been processed by the meter. The counter is updated for each packet.
3. Meter Bands works in the following way:

- The meter that measures the rate of each incoming attached flow.
- Holds the instructions and executes the operations based on the measured rate of the flows.
- Every Meter band carries instructions on what to do when a flow reaches a specified set rate.
- Applies actions when the flowrate is greater than the specified set rate of the meter [24].

A meter can define numerous meter bands, with only one Meter band being be applied each time the packet passes through the meter. In an event whereby a meter has many Meter bands defined, the Meter band with the highest set rate still being below the current measured flow rate takes priority in execution. In an event whereby the flowrate is lower than any of Meter band rates configured, no actions are executed.

A Meter band fundamentally has two basic actions that it can perform when a flow exceeds the set rate of the band. The two actions are the:

- Drop - Orders the Meter band to drop every packet if the current flow rate exceeds the Meter bands rate.
- DSCP Remark - allows the Meter band to increase the drop rate by modifying the 6-bit DSCP field. In the AF standard, this can be done by increasing the DSCP value while still in the same class. With DSCP remarking applied based on current flow rate, a DiffServ domain could be created.

In summary, a Meter measures the rate and forwards that information to multiple Meter bands that apply actions based on that rate. A meter entry is attached to a flow that directs the flow into a specified Meter. Every meter entry consists of the Meter table.

The ability to change DSCP values based on flow rates, higher priority can be assigned to flows within given rates and lower the flows priority if they exceed it. This ensures that bandwidth guarantee can be achieved. The lower priority traffic is only dropped in cases of congestion, excess traffic is allowed when enough network resources are available to meet the QoS requirements of traffic flows.

The priorities were assigned as follows:

- PAF = AF traffic priority = 2.
- PEF = EF traffic priority = 3.
- PBE = BE traffic priority = 1.

If the length of a queue class is longer than other queues, that queue will be served faster than the queue whose length is smaller. If the queues are of equal length, they will be served according to their priority value. Each traffic class therefore is serviced as per the product of queue length and assigned priority. For example, traffic in EF class will be serviced fastest when its queue length is large. Traffic in AF class will be serviced faster only if when its queue length is higher. BE traffic will serviced highest when there is no AF and EF class traffic.

4.4 Proof of Concept

In the proof of concept (PoC), the objective is to prove the applicability and validate the methods and theories of QoS mechanisms in SDN using real-time tests and experiments. A PoC has been

established to offer and back with evidence that SDN can achieve bandwidth guarantees through QoS with the help of DSCP and Meters.

Virtual switches are used in implementing of PoC through emulation. An emulated network environment with hosts and SDN-compatible switches meets the requirements to perform different tests to prove and validate the concepts, theories and methods. The basic requirements for the network environment are that it should offer support for SDN-switches and is OF compatible. This allows for the usage of SDN controllers which is required to offer DSCP remarking with Meters. The suitable tool for implementation of these concepts is a networking emulation software called Mininet.

4.4.1 Development Tools

Experimentation apparatus is revealed in this section and was used to setup the different network topologies for a design research goal. The tools will be discussed in this chapter.

4.4.2 Mininet

Mininet provides the following advantages:

- Mininet is an open source network emulator and its core focus is on OF and SDN-controllers that are OF compatible.
- Mininet is a tool for emulating SDN networks.
- Mininet runs on a Linux kernel and uses Python API.
- It offers a complete networking experience.
- It provides all the network functionalities from end-hosts to switches and routers.
- Mininet tool can be used to emulate a complete network setup.

- Virtual hosts, switches, links, and controllers are an equivalent version of those implemented in hardware.
- Allows for the creation of a customized network of choice.

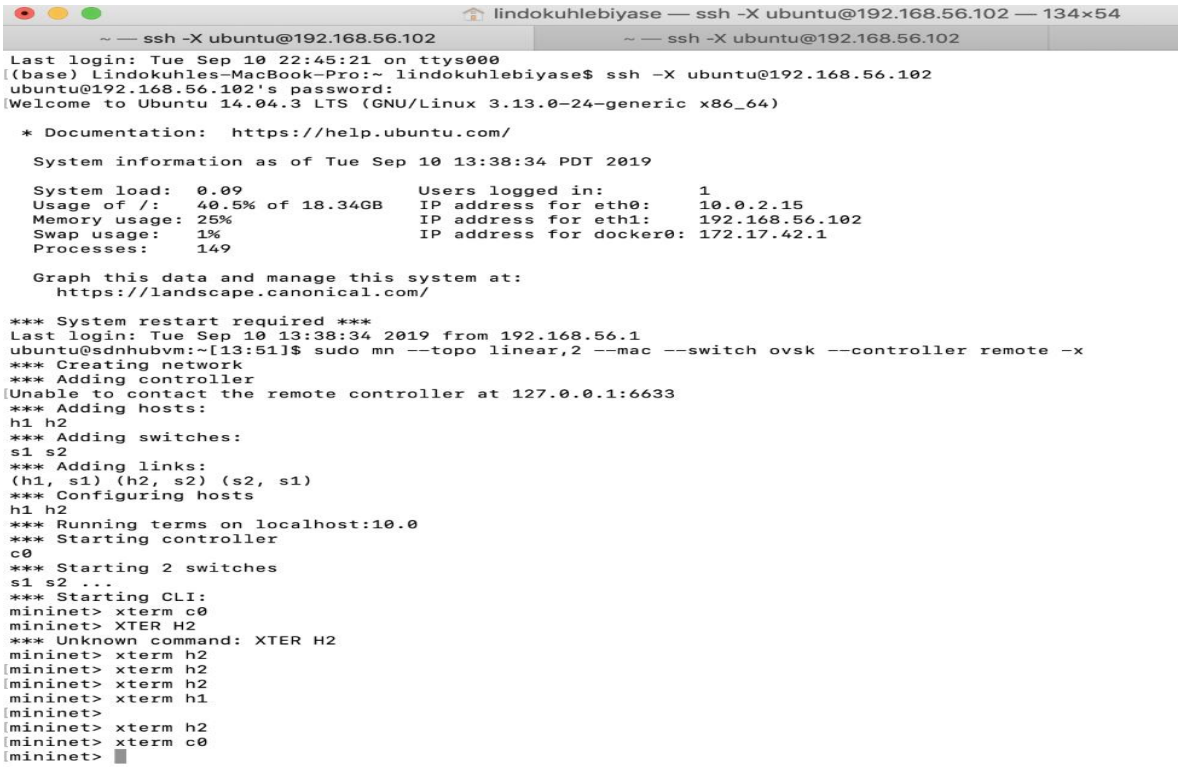


Figure 4.4.2 Mininet Invocation Environment

An emulated SDN network can be configured to behave in a similar way as the real SDN network. An emulated SDN network is created and is considered sufficient to create a proof of concept. Mininet imposes limitations compared to a real physical network. These limitations include:

- All Mininet components share the same computer resources.
- Sharing of resources creates competition and results into a slow experience than the physical network can offer.

- It is not suited for experiments ranging around 10 Gbps.

As a trade-off for performance, emulated software has the advantage of being customizable. It can create topologies based on the user's choice, instead of being limited by the number of switches purchased, physical space, etc.

Mininet is used to implement the PoC and gives supporting evidence that DSCP remarking with meters can be used to enhance QoS provisioning. The ability to customize network allows more test cases than a high speed network.

Recommended resources for Mininet network emulation are given below:

- Mininet requires Linux Operating System (OS) to run.
- Ubuntu is the recommended distribution to use.
- Using a Virtual Machine (VM) is recommended for the use of Mininet, thus running Mininet inside an OS that is provided from the VM [25].

Benefits of using a VM for running Mininet experiments are as follows:

- VM allows for virtualization of an OS of choice that provides the best compatibility with Mininet.
- VM provides benefits such as changing OS and OS version with ease and modifying the OS according to desired preferences, without impacting the host computer.

4.4.3 Oracle VM Virtual Box

The chosen VM used is the VirtualBox. VirtualBox is the recommended choice of VM from Mininet [25]. The key factor that was taken into consideration is that the VirtualBox is free

compared to VMware which is a popular VM that only has a free trial requires subscription to keep on using its services. [26].

A VM is an emulator. An emulator can emulate different systems. Emulation enables the computer system to imitate another computer system. This enables the computer to run programs and tools designed for other computer systems [27]. This means that any OS can be used to run Mininet. The VM can emulate any OS for Mininet without restrictions on what native OS the computer runs on.

VMs such as VirtualBox creates a separate environment for the emulated OS to run on. The emulated environment cannot infringe on the Host OS allowing the user to use multiple OS simultaneously, in different windows.

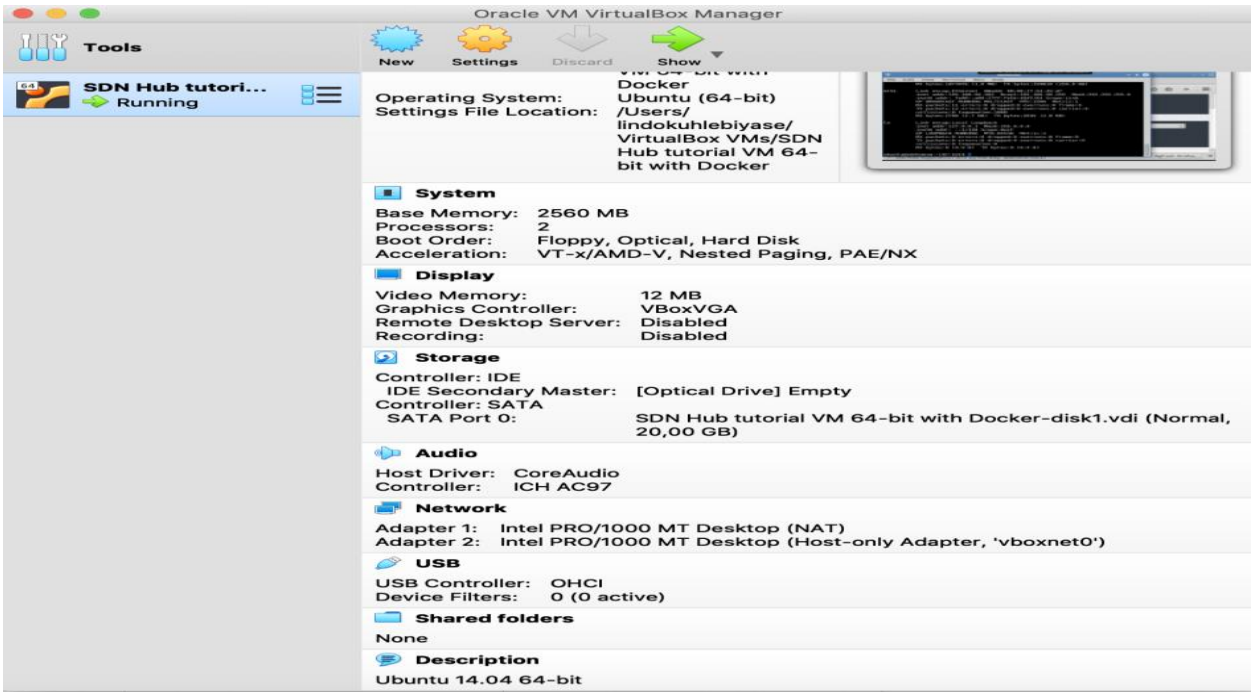


Figure 4.4.3: VirtualBox Environment

4.4.4 SDN Hub

SDN Hub provides an OS with Mininet already configured, installed and several other tools and controllers which are useful. The SDN Hub OS is Ubuntu which is compatible and recommended for use with Mininet. Additional tools for SDN developing are included which provide ease-of-use to the user that does not have to install these tools separately.

Tools that come with SDN Hub include:

- SDN Controllers: OpenDaylight, ONOS, Ryu, Floodlight, FloodlightOF1.3, POX, and Trema.
- Open vSwitch 2.3.0 with support for OF 1.2, 1.3 and 1.4
- Mininet
- Pyretic
- Wireshark 1.12.1
- JDK 1.8, Eclipse Luna, and Maven 3.3.3.

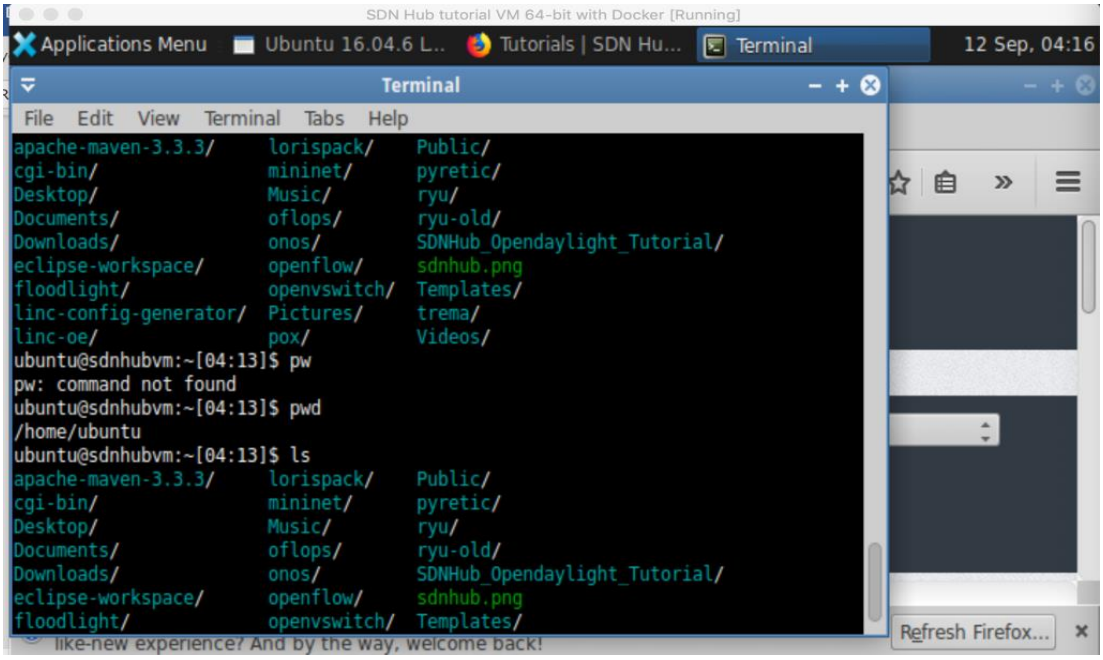


Figure 4.4.4: SDNHub Environment with Tools

A preconfigured VM with Mininet installed with useful tools offers a huge ease-of-use advantage over creating a fresh VM with Ubuntu and manually installing required tools. The benefits result in the choice of SDN Hubs preconfigured VM over creating a new fresh Operating System from scratch.

SDN Hubs VM provides the virtual SDN switch Open vSwitch with the following features:

- Open vSwitch for usage with Mininet.
- Open vSwitch (OVS) is arguably the most used virtual SDN switch.
- OVS supports OpenFlow 1.3.
- It does not have any support for Meters.

Meters plays a critical role in how QoS and Bandwidth guarantee will be achieved. Support for Meters are planned in release 2.8 [28]. Therefore, another virtual SDN switch must be used.
The Openflow Soft switch (OfSoftswitch) is a virtual SDN switch which provides support for the Meters that OVS does not support. OF 1.3 and Meters is all that is required to achieve QoS [29]. OFSoftSwitch is compatible with Mininet as the emulated network environment chosen [30].

4.4.5 Controller Choice

The tools needed for testing and running PoC experiments, involves the controller that should be chosen to manage the network. The selection of the controller is derived from the project requirements and objectives.

In order to achieve QoS the following requirements must be satisfied:

- DSCP will be used.
- Meters will implement DSCP remarking.
- The switches need to support OpenFlow 1.3 for Meters to be enabled.
- Controllers must support OpenFlow 1.3, otherwise they won't be compatible with each other.
- A controller suited for quick deployment and prototypes is suitable for the purpose.
- Ryu is the best controller for OpenFlow support as it is suitable for quick prototyping [31], [32].

With all the needed tools available, the PoC experiments are conducted and the theories and methods will be tested.

5. Proof of Concept Tests

Experiments are performed in an emulated environment that behaves like a physical SDN network to validate the PoC. With this, it can be further researched if SDN can implement bandwidth management and provide QoS guarantee.

To conduct these PoC tests, a test topology is required. It is a basic topology with the bare requirements to test QoS in SDN networks. The test topology of the network must be able to simulate congestion to see that different priority levels are working as desired.

5.1 Solution Design

This section presents the chosen SDN topologies that will be used to test the concept of bandwidth scalability management and flow aggregation.

5.1.1 Solution Design Process

Bandwidth management and flow aggregation are amongst the core issues which must be solved when scaling up medium to large network enterprises. Network components have limited computing resources which must be shared among different network activities such as routing, rerouting, measurements, security, link discovery, path recovery and load balancing. Due to the complexity in managing the bandwidth when the network grows, it gives rise to a scalability concern that needs to be addressed.

The scalability of bandwidth management and flow aggregation in SDN can be addressed using different methods. It is essential to design a solution to improve scalability in SDN. The aim of this thesis is to investigate the scalability of bandwidth management and flow aggregation techniques in SDN.

The following flowchart in Figure 5.1 is a summary of the design process.

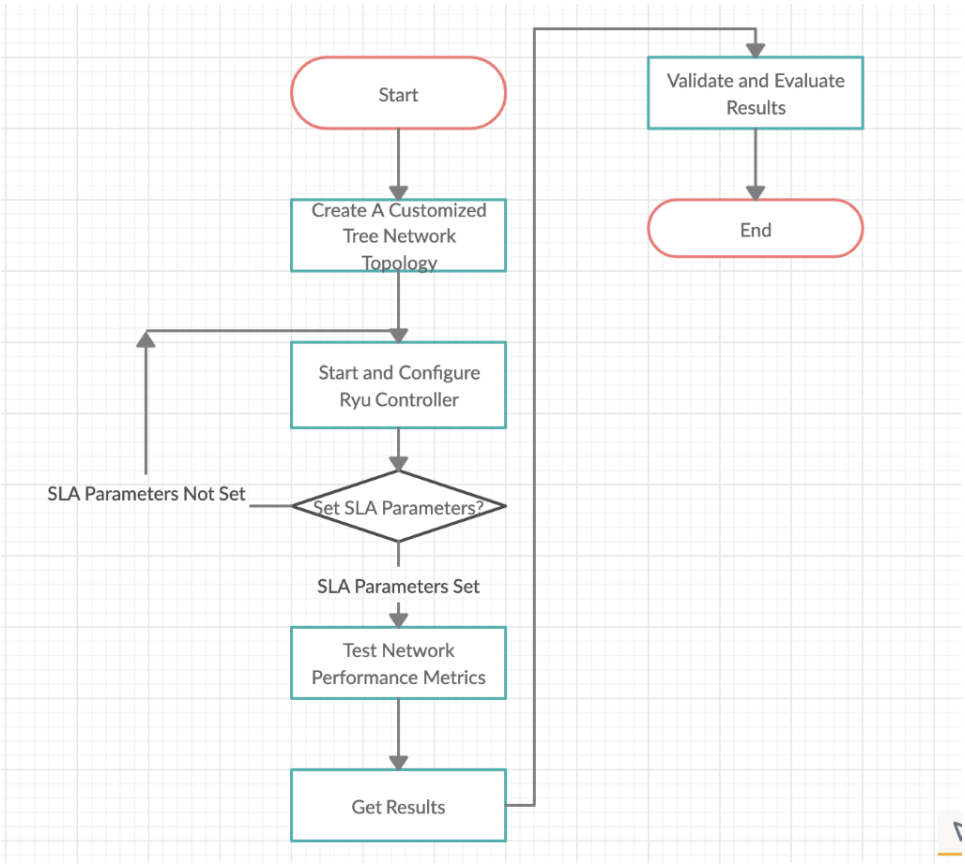


Figure 5.1.1 Design to Validation and Evaluation of Results

5.2 Solution Design Models

This section introduces network topology used for investigating scalability of bandwidth management and flow aggregation methods applied to SDN.

5.2.1 Tree Topology Model: Network Performance

The model is used to investigate the scalability of flow aggregation and bandwidth management methods in SDN.

- Scalability of bandwidth management and flow aggregation is considered as a complex problem that needs to be attended by decomposing the problem under which the network is most probably expected to perform.
- Tree Network Topology is made up of 7 switches and is shown in the following figure 5.2.1 as the SDN topology.
- The model has been chosen for a constant number of network nodes with 16 hosts while testing the bandwidth management scalability under the given condition.

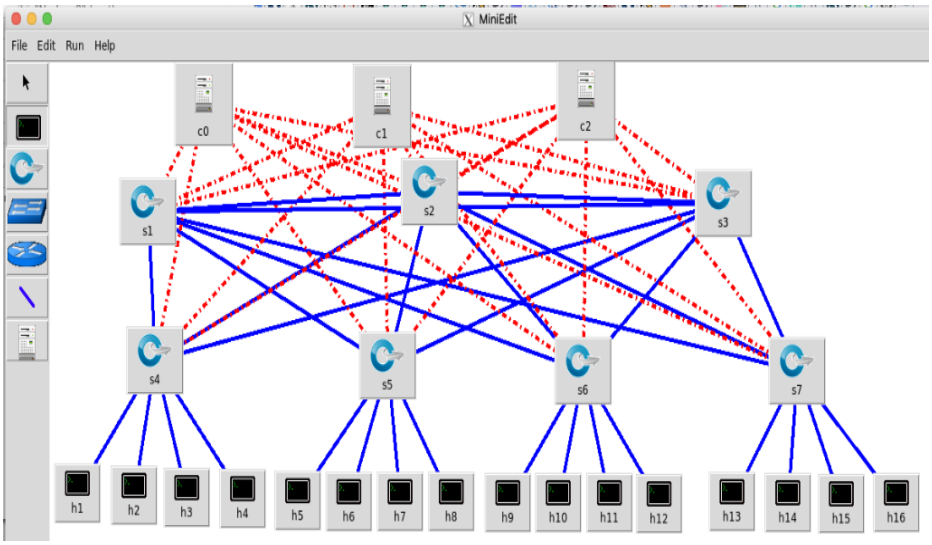


Figure 5.2.1. Customized Tree Topology with 16 Hosts

- The controller controls and communicates with all 7 switches via the OpenFlow protocol.
- The Tree Topology was chosen to effectively prove bandwidth management and flow aggregation acceleration under constant network nodes test.
- Hosts are labelled from h1 to h16 and under different scenarios the test will be conducted between any two hosts to prove the concept.

- The routes are chosen to include the maximum number of switches in 2 different use case scenarios.

Scenario 1

- Per flow QoS operation

Scenario 2

- QoS by using DiffServ operation

The results obtained using these scenarios will be compared to prove the scalability of bandwidth management and flow aggregation in SDN.

5.3 Scalability Bandwidth Management Models

This section outlines the selection of each model and provides the rationale behind the choice of each model in each of the use case scenarios. QoS is a technology that can transfer the data in accordance with the priority based on the type of data, and reserve network bandwidth for a communication in order to communicate with a constant communication bandwidth on the network.

5.3.1 Model Setup Description

The purpose of this thesis is to prove the case of scalability of bandwidth management and flow aggregation in SDN, as one of the measures of enhancing the scalability of the network. The core focus is then on the bandwidth management techniques that guarantees QoS applied to the switching elements in the data plane.

The Model is designed to test the maximum number flows with SLA and QoS parameters through each switch using the preferred route.

Model 1 setup is a network topology to allow maximum flow through a chosen path between two hosts and the controller can oversee a maximum of 7 switches as depicted in the diagram, 16 hosts are connected to 7 switches of the SDN topology.

The test setup summary is given below:

- Mininet running on Ubuntu 18.04.1 LTS
- The Ryu Controller running on Python.
- Network access for the switch and controller communication.
- Python Programming Language.
- OpenFlow Switch.

The customized network topology is designed using Python programming language and it is imported into Mininet to conduct experiments. The topology is created with queue settings and rules to reserve network bandwidth and allows for traffic shaping. The communication link is designed to have 1Mbps capacity to prevent congestion in the link when large packets are being sent to and from the server. The objective is to study the throughput of this network topology.

A network topology is a graph G = (V, E) with capacities c(u, v) for every edge (u, v) ∈ E. Among the nodes V are hosts, which send and receive traffic flows, connected through non-terminal nodes called switches. Each host is connected to one switch, and each switch is connected to zero or more hosts, and other switches. For switch-to-switch edges (u, v), the capacity c(u, v) = 1 is set, while host-to-switch links have infinite capacity. This allows for test the stress of the topology. A traffic matrix (TM) defines the traffic demand: for any two hosts v and w, T(v, w) is an amount of requested flow from v to w.

The throughput (T) of a network G with TM is defined as the maximum value t for which T ·t is feasible in G. This means that the maximum t for which there exists a feasible multi commodity flow that routes flow T(v, w)·t through the network from each v to each w, subject to the link capacity and the usual flow conservation constraints. This can be formulated in a standard way as a linear program and is thus computable in polynomial time. If the nonzero traffic demands T(v, w) have equal weight, this is equivalent to maximizing the minimum throughput of any of the requested end-to-end flows. This maximization is a standard problem called maximum concurrent flow [50]. Furthermore, this objective function captures the notion of fairness among flows

5.4 Implementation

The solution designed is implemented on Mininet using Python programming language to prove the scalability of bandwidth management and flow aggregation which is the core focus of the project.

5.4.1 Solution Design Implementation

This section outlines the design implementation procedure to realize the network topology. Python programming language is used to create the topology for conducting the experiments and testing of the network topology using Mininet. For both models the Flowchart below will be used, and results will be compared, analyzed and evaluated in the next chapter.

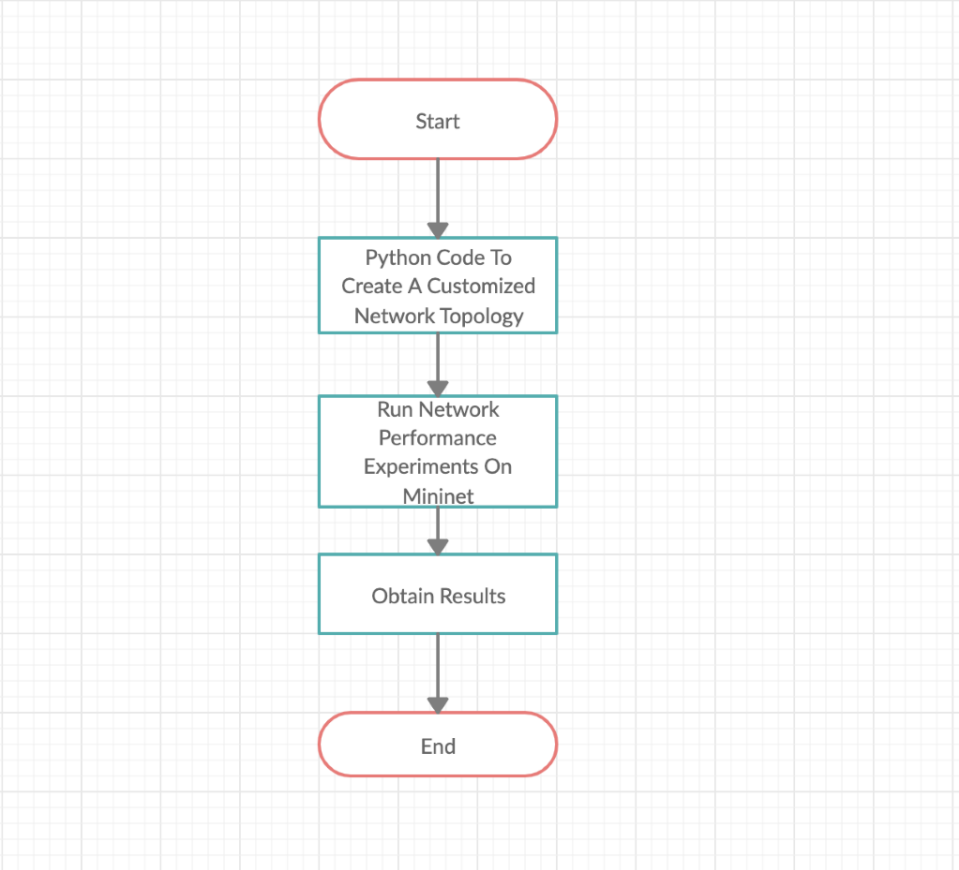


Figure 5.4.1(a) Implementation Flow Chart.

- The customized tree topology is created in Python with parameters configured in the code, parameters such as line loss, bandwidth, and delay.
- The code then is imported into Mininet where the design code is emulated by Mininet and some testing are conducted to observe topology performance metrics.
- The SDN controller runs remotely and it facilitates communication between the switch and the controller via OpenFlow protocol.
- The controller that is used for the experimentation is the Ryu controller and the Code for the Ryu controller is found in the Appendix.

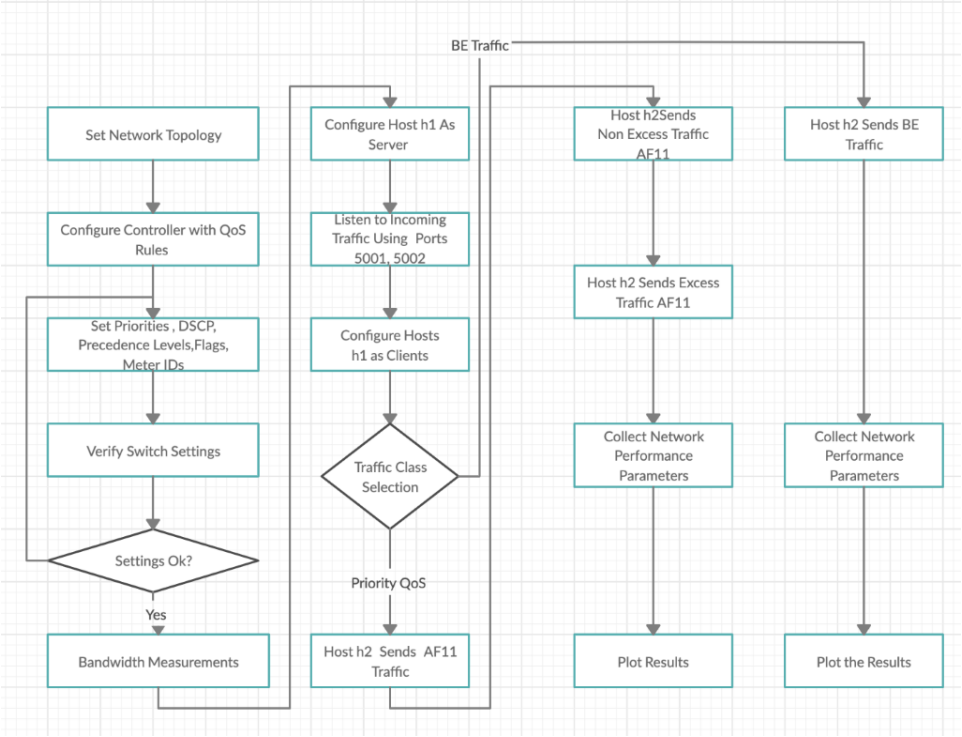


Figure 5.4.1(b) Model Set up Implementation Flowchart.

5.4.2 Per Flow QoS Operation Model Emulation

This section outlines the procedure undertaken when emulating the network topology depicted in Figure 5.2.1. In this emulation model, the throughput, jitter, datagram transfer rate and datagram percentage losses are observed. Through using Mininet commands a client-server architecture is setup between the two hosts for the observing the SLA and QoS parameters during the test.

The following Figure 5.4.2(b) depicts the procedures undertaken in setting up the client-server relationship between the two hosts. Mininet prompts appear when the network has been created, the network topology has 16 hosts, and any two hosts can be chosen to run testing commands. The host's windows are invoked by using the ‘xterm h2 h1’ command in the Mininet prompt with the two end hosts h2, h1 being selected to test the network.

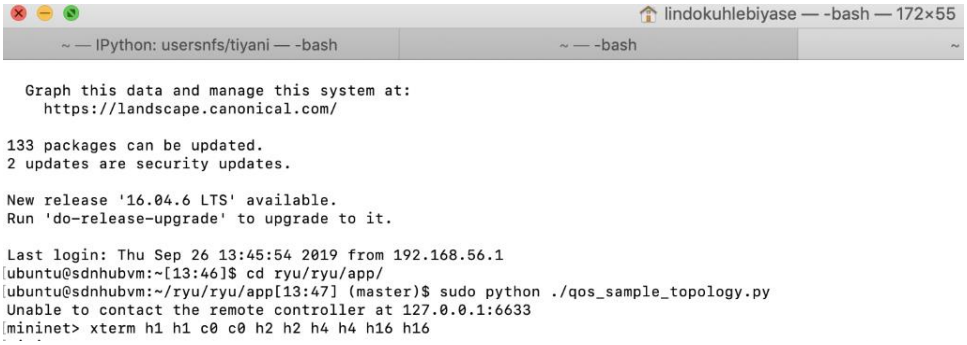


Figure 5.4.2(a) Mininet Command invoking Network Nodes including Hosts h1 and h2.

The throughput capacity of the network is an important parameter of the network. It indicates how reliable the network and it indicates the loss probability of packets which can be dropped by the network. In testing the throughput of the network, two hosts from the network end-users where selected. The following Figure 5.4.2(b), (c) shows where one host was set up as the server and another one as the client.

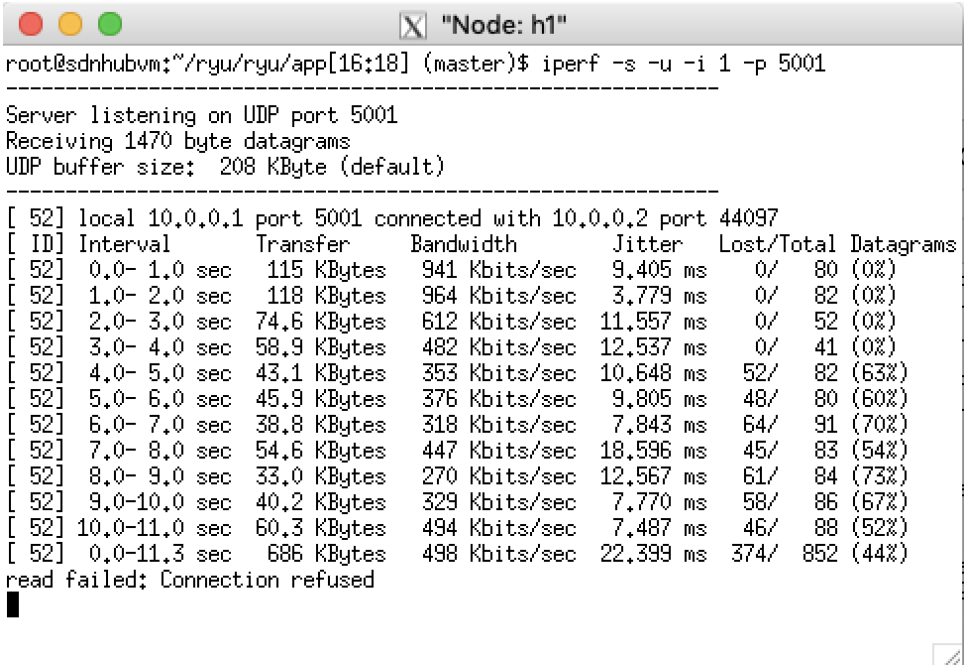


Figure 5.4.2 (b) Server Setup Listening on UDP Port 5001

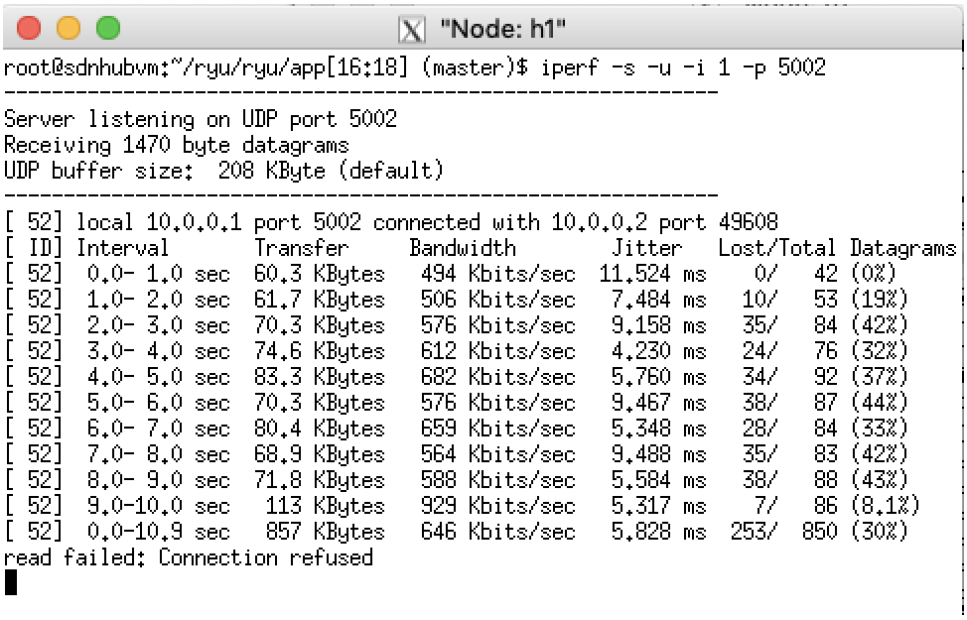


Figure 5.4.2. (c) Server Setup Listening on UDP Port 5002

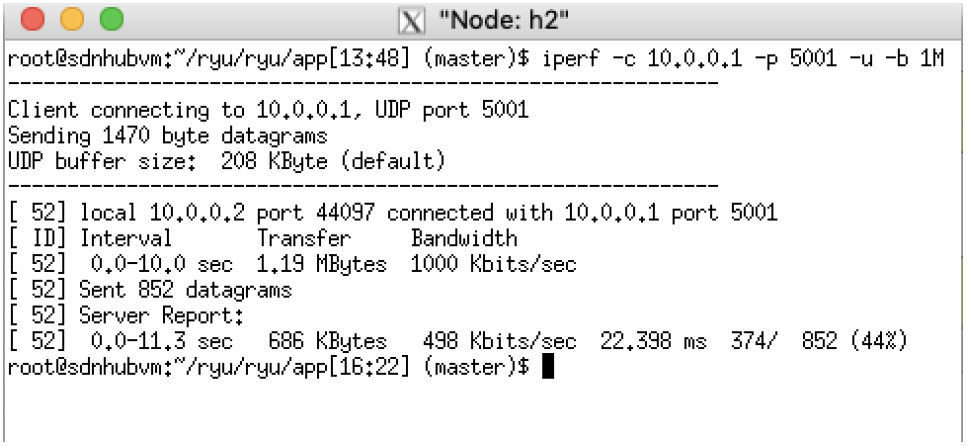


Figure 5.4.2. (d) Client Setup Sending Traffic through Port 5001

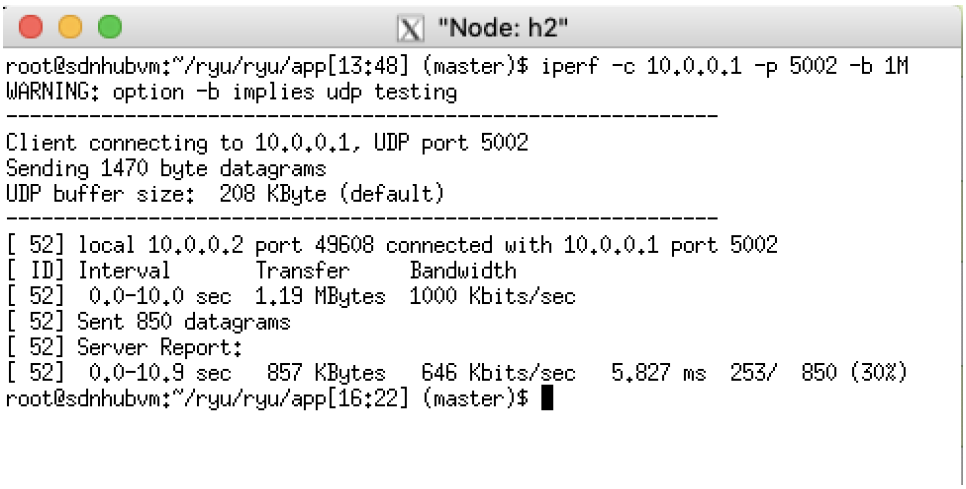


Figure 5.4.2. (e) Client Setup Sending Traffic through Port 5002

The Client-Server architecture was set up in the following way:

- Host h2 was set up to be the client node sending UDP Traffic through ports 5001 and 5002.
- Host h1 was set up to be the server node listening to incoming traffic on ports 5001 and 5002.
- Network packets are sent from the client to the server nodes and the measurement of network throughput is done.
- The server is setup using the Mininet command: ‘iperf -s –u -i 1 -p 5001’.
- The client is setup using the Mininet command: 'iperf -c 10.0.0.1 -p 5001 –u –b 1M'.
- The test was performed for this model with results and analysis found in section 6.1.
- The switches were configured for OpenFlow1.3 protocol and to implement traffic shaping to prevent network congestion.

5.4.3 Per Flow QoS Operation Controller Configuration

The controller is configured to:

- Implement the packet processing and switching.

- Modify flow table pipeline processing for registration of flow entries.
- The setup is shown in Figure 5.4.3 below.

Figure 5.4.3(a) Installing Flow Entries in the switches

- Configured switches with QoS settings depicted on Figure 5.4.3 (b), (c).

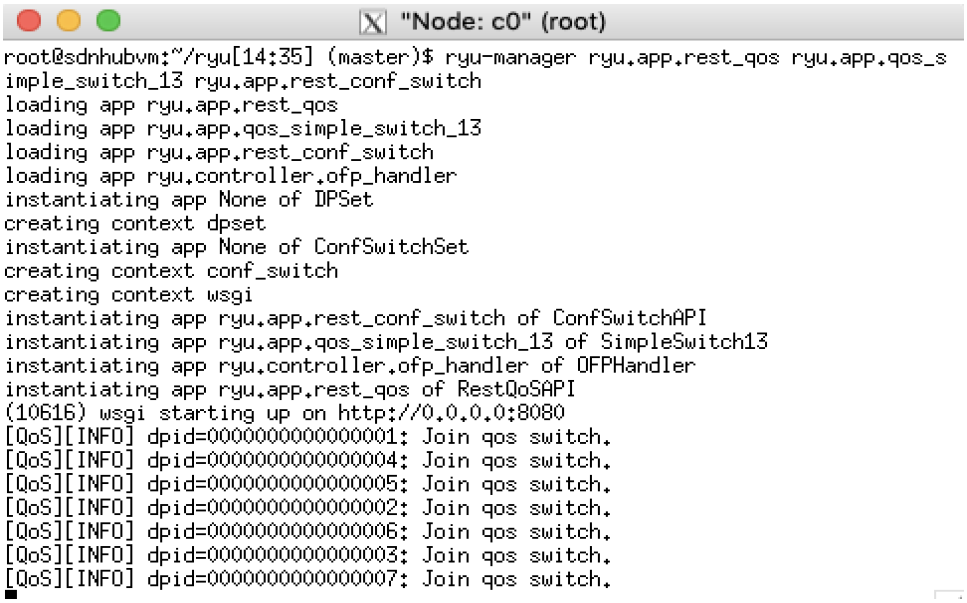


Figure 5.4.3(b) Setting QoS Settings for the switches in the Controller.

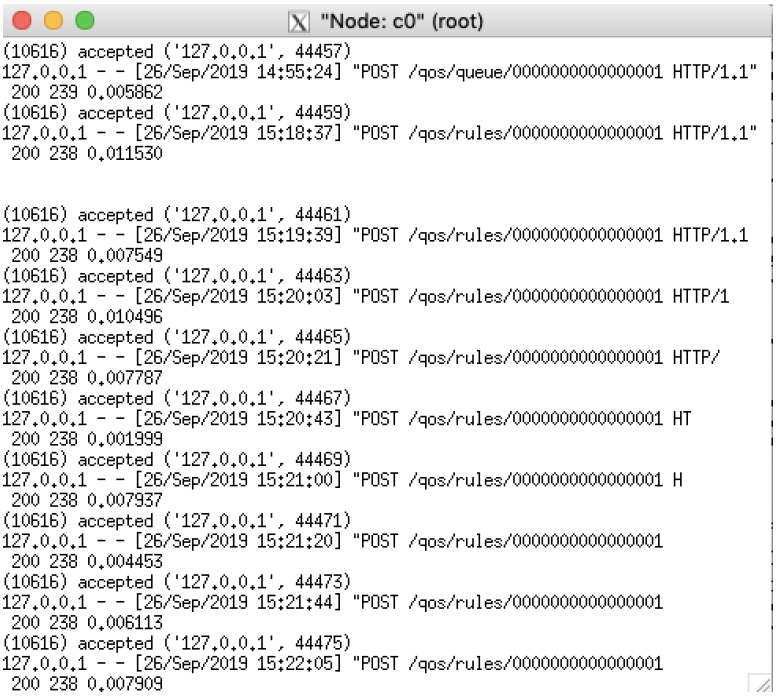


Figure 5.4.3(c) Setting QoS Settings in the for the switches in the Controller

The queue settings on the switches were set as shown in the Table:

Queue ID	Max Rate	Min Rate
0	500 Kbps	-
1	1000 Kbps	800 Kbps

Table 6: Depicts the Switch queue Settings

5.4.4 Bandwidth Measurement: Per Flow QoS Operation

Bandwidth measurement was conducted between the client hosts h2 and the server hosts h2 in the following way:

- Host h1 (server) listens on ports 5001 and 5002 with UDP protocol with the files saved.
- Host h2(client) sends 1Mbps UDP traffic to the port 5001 on h1 and 1Mbps UDP traffic to the port 5002 on h1.
- The server host h1 and client host h2 are shown in Figure 5.4.4(a), (b), (c), (d) below:

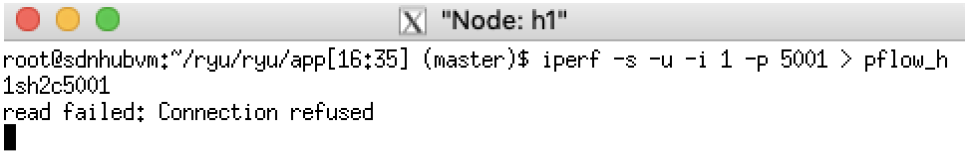


Figure 5.4.4(a) Server Host h1 Listening on Port 5001

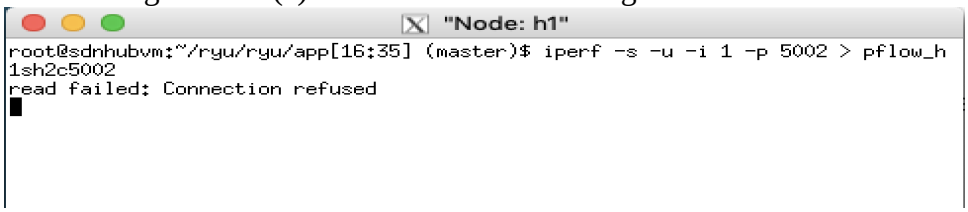


Figure 5.4.4(b) Server Host h1 Listening on Port 5002

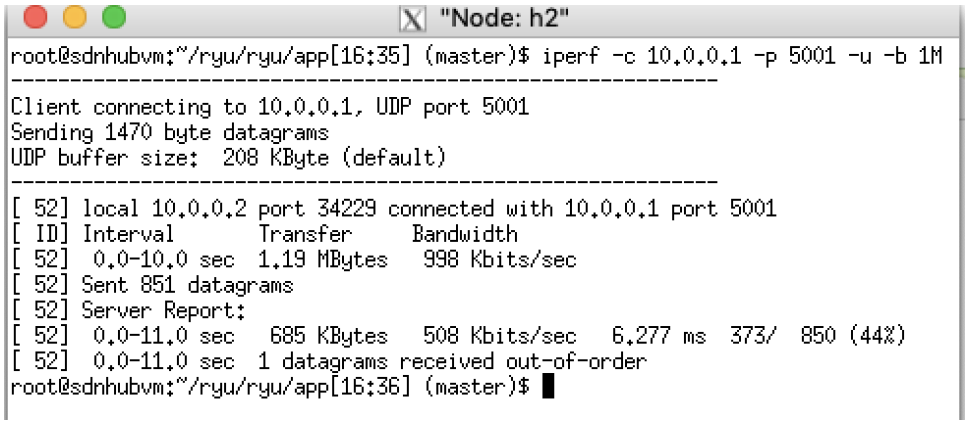


Figure 5.4.4(c) Client Host h2 Sending UDP Traffic using Port 5001

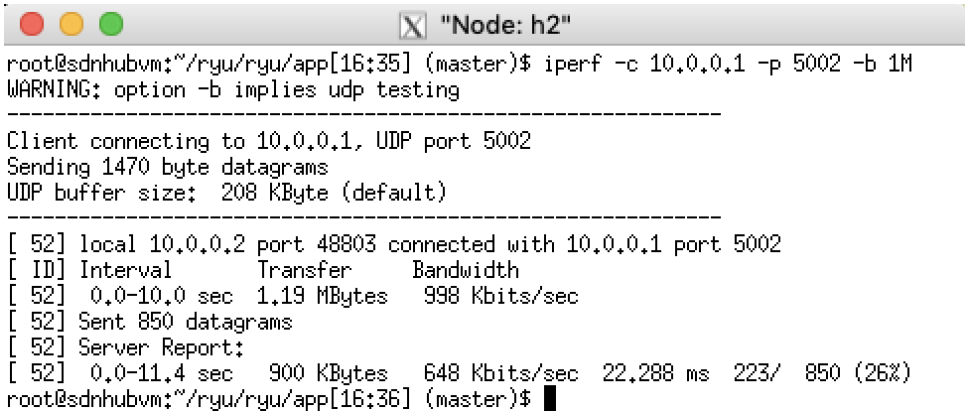


Figure 5.4.4(d) Client Host h2 Sending UDP Traffic using Port 5002

- The UDP Traffic statistics were saved on the file and its contents could be retrieved using 'more pflow_h1sh2c5002' and 'more pflow_h1sh2c5001' commands as shown in Figure 5.4.4(e), (f) below:

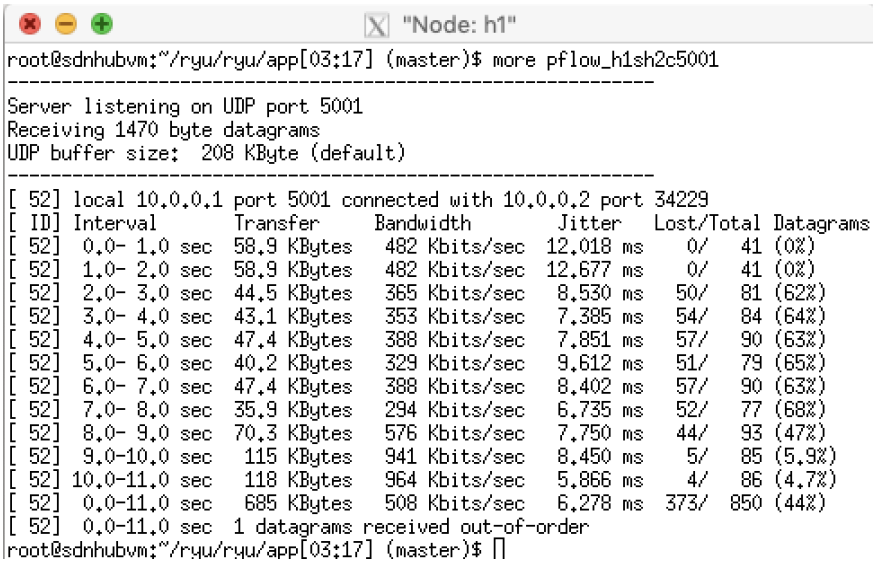


Figure 5.4.4(e) Retrieving the Server Host h1 Port 5001 UDP Traffic file contents.

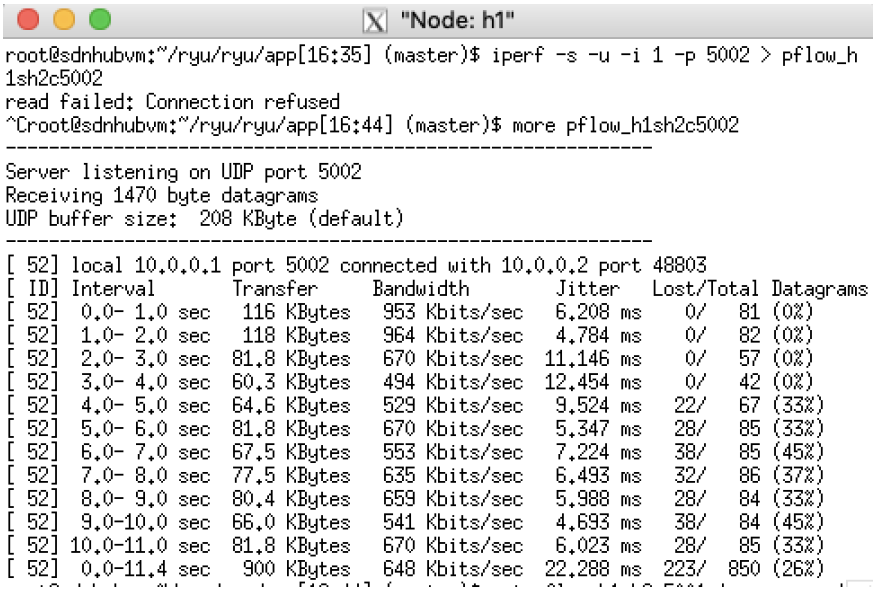


Figure 5.4.4(f) Retrieving the Server Host h2 Port 5002 UDP Traffic file contents.

5.5 Scalable QoS by DiffServ Operation Model

This model divides flow into the several QoS classes at the entrance router of DiffServ domain and applies DiffServ to control flows for each class. DiffServ forward the packets according to PHB defined by DSCP value which is the first 6-bit of ToS field in the IP header, and realizes QoS. This model is emulated using a customized linear topology model (shown in Figure 5.5.1(a)) which scales bandwidth management with relative ease compared to the tree network topology.

This model includes operations such as setting queue and bandwidth configuration based on the QoS class into Switch (Router) and installation rules of marking the DSCP value in accordance with the flow.

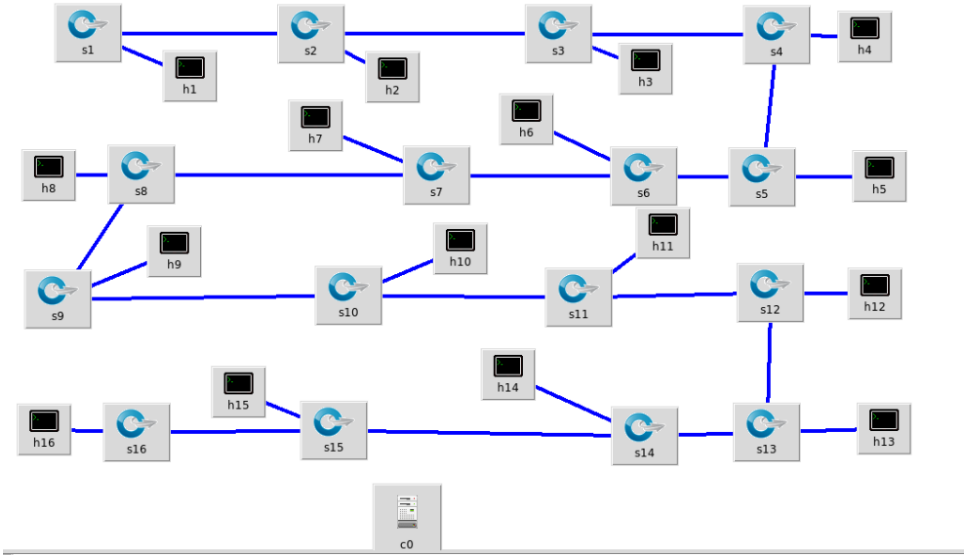


Figure 5.5.1(a) Customized Linear Topology with 16 Hosts

This section outlines the procedure undertaken when emulating the network topology depicted in Figure 5.5.1(a). In this emulation model, the throughput, jitter, datagram transfer rate and datagram percentage losses are observed. Through using Mininet commands a client-server architecture is setup between the two hosts for the observing the SLA and QoS parameters during the test.

The following Figure 5.5.1(c) depicts the procedures undertaken in setting up the client-server relationship between the two hosts. Mininet prompts appear when the network has been created, the network topology has 16 hosts, and any two hosts can be chosen to run testing commands. The host's windows are invoked by using the 'xterm h2 h1' command in the Mininet prompt with the two end hosts h2, h1 being selected to test the network.

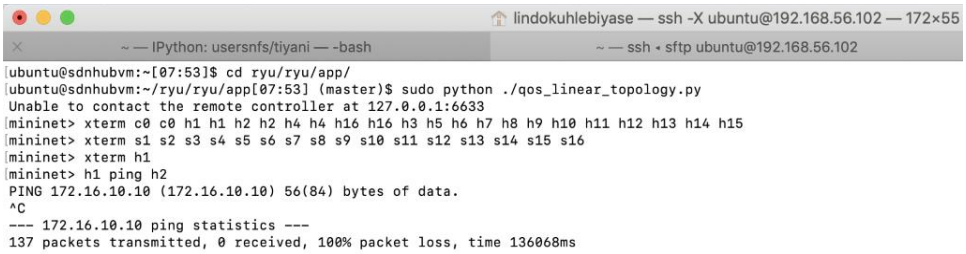


Figure 5.5.1(b) Mininet Command invoking Network Nodes including Hosts h1 and h2.

The throughput capacity of the network is an important parameter of the network. It serves as an indicator of network reliability and it indicates the loss probability of packets which can be dropped by the network. In testing the throughput of the network, two hosts from the network end-users where selected. The following Figure 5.5.1(c), (d) shows where one host was set up as the server and another one as the client.

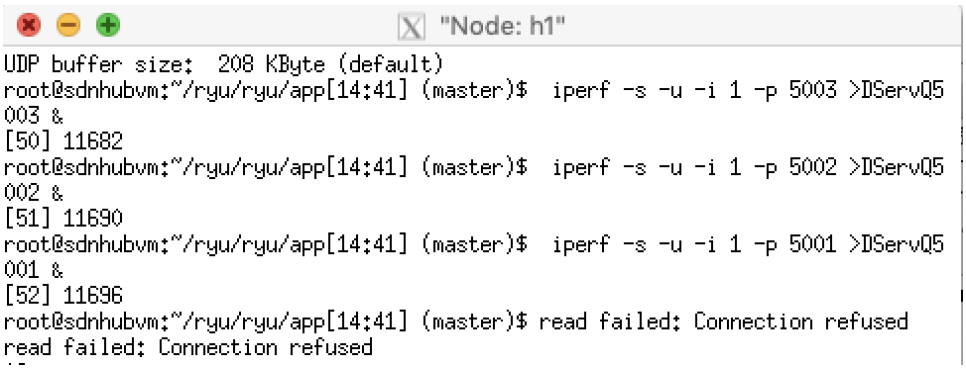


Figure 5.5.1 (c) Server Setup Listening on UDP Ports 5001, 5002 and 5003

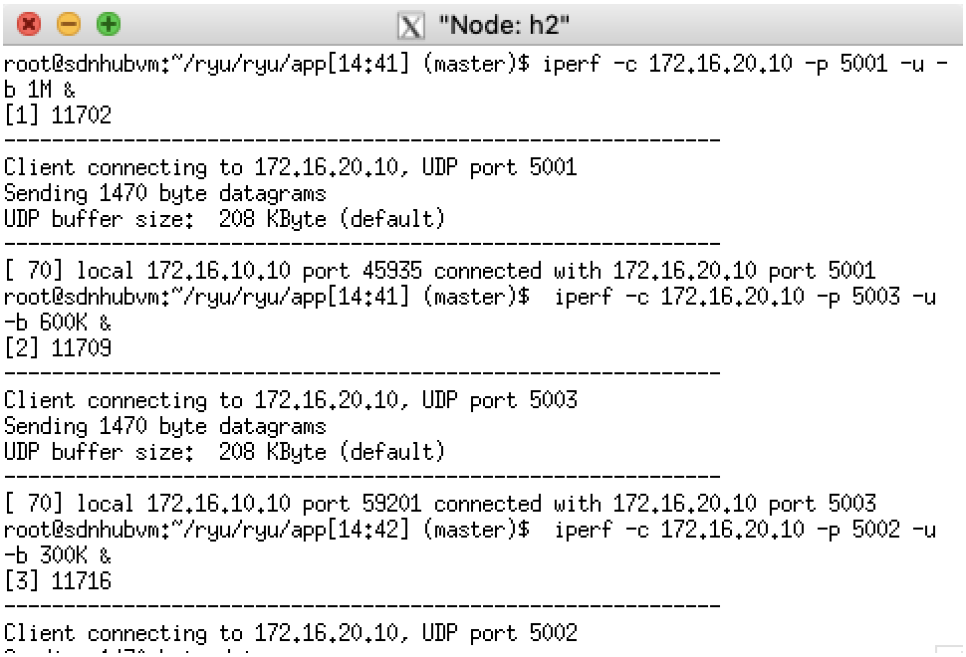


Figure 5.5.1 (d) Client Setup Sending Traffic through Ports 5001, 5002, and 5003

Client-Server architecture was set up in the following way:

- Host h2 was set up to be the client node sending UDP Traffic through ports 5001, 5002, and 5003.
- Host h1 was set up to be the server node listening to incoming traffic on ports 5001, 5002, and 5003.

- Network packets are sent from the client to the server nodes and the measurement of network throughput is done.
- Server node h1 is setup using the Mininet command: ‘iperf -s –u -i 1 -p 5001’.
- Client node h2 is setup using the Mininet command:'iperf -c 10.0.0.1 -p 5001 –u –b 1M'.
- The test was performed for this model with results and analysis found in section 6.1.
- The switches were configured for OpenFlow1.3 protocol and to implement traffic shaping to prevent network congestion.

5.6 Setting up IP Addresses

The automatically assigned IP addresses were deleted and new IP addresses for all the 16 hosts represented in Figure 5.5.1(a) were assigned as depicted in the table below:

Host	Assigned IP Address	Interface	Default Gateway IP
h1	172.16.20.10/24	dev h1-eth0	172.16.30.10
h2	172.16.1010/24	dev h2-eth0	172.16.30.10
h3	172.16.30.10/24	dev h3-eth0	172.16.30.10
h4	172.16.40.10/24	dev h4-eth0	172.16.30.10
h5	172.16.50.10/24	dev h5-eth0	172.16.30.10
h6	172.16.60.10/24	dev h6-eth0	172.16.30.10
h7	172.16.70.10/24	dev h7-eth0	172.16.30.10
h8	172.16.80.10/24	dev h8-eth0	172.16.30.10
h9	172.16.90.10/24	dev h9-eth0	172.16.30.10
h10	172.16.100.10/24	dev h10-eth0	172.16.30.10
h11	172.16.110.10/24	dev h11-eth0	172.16.30.10
h12	172.16.120.10/24	dev h12-eth0	172.16.30.10
h13	172.16.130.10/24	dev h13-eth0	172.16.30.10
h14	172.16.140.10/24	dev h14-eth0	172.16.30.10
h15	172.16.150.10/24	dev h15-eth0	172.16.30.10
h16	172.16.160.10/24	dev h16-eth0	172.16.30.10

Table 7: IP Address Interfaces Assigned to hosts in the Network

Figure 5.6.1(a), (b) depict the configuration command used to delete the automatically assigned an IP address and the command to assign a new IP address:

- Command to delete IP address: ip addr del 10.0.0.1/8 dev h1-eth0.
- Command to assign new IP address: ip addr add 172.16.20.10/24 dev h1-eth0.

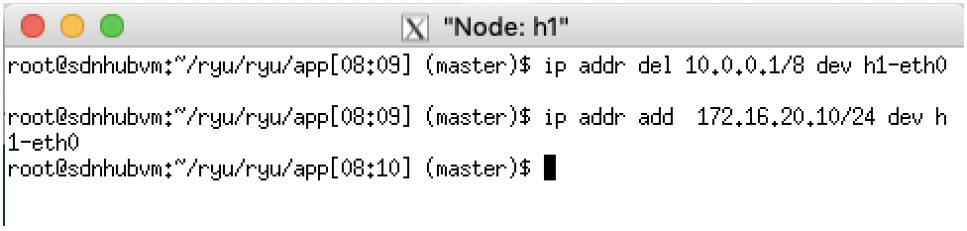


Figure 5.6.1(a) Assigning new IP address to Host h1

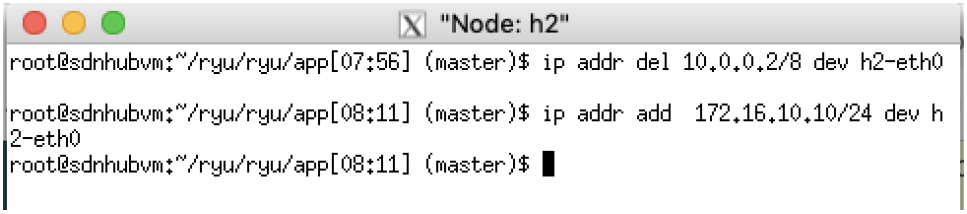


Figure 5.6.1(b) Assigning new IP address to Host h2

5.6.2 Scalable DiffServ QoS Operation Controller Configuration

The controller is configured to:

- The network topology script: qos_linear_topology is invoked in terminal environment.
- The script implements a linear network topology with 16 switches and 16 hosts.
- The controller is configured to load the following QoS settings depicted in the table below, on all the switches.
- Implement the packet processing and switching.
- Modify flow table pipeline processing for registration of flow entries.

- The setup is shown in Figure 5.6.2(a), (b), (c) below.

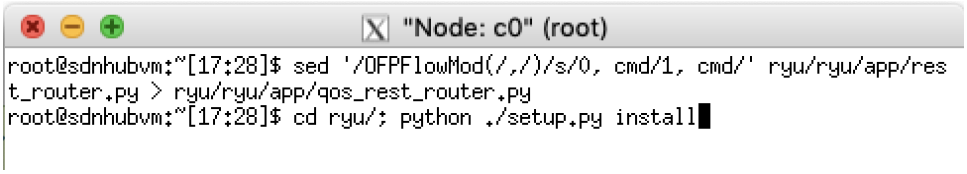


Figure 5.6.2(a) Showing the Controller Set up Commands

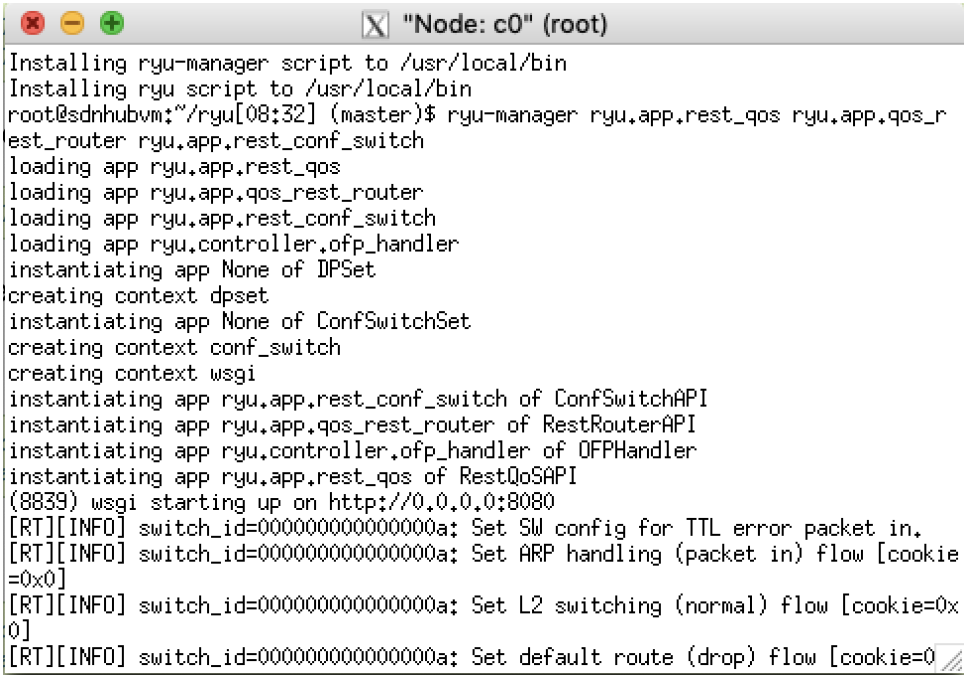


Figure 5.6.2(b) Router configuration commands on the Controller

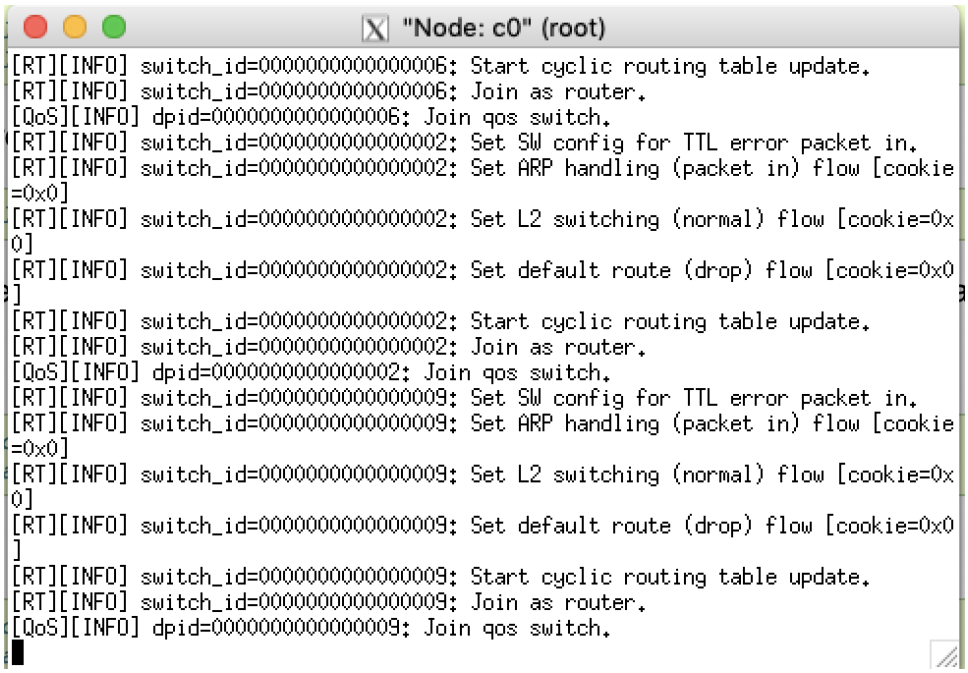


Figure 5.6.2(c) Showing the router features on the Controller

5.6.3 Scalable DiffServ QoS Queue and Router Settings

- Queue settings were configured as displayed in the table below:

Queue ID	Maximum Rate	Minimum Rate	Class
0	1 Mbps	-	Default (BE)
1	1 Mbps	200 Kbps	AF31
2	1 Mbps	500 Kbps	AF41

Table 8: QoS Queue Settings

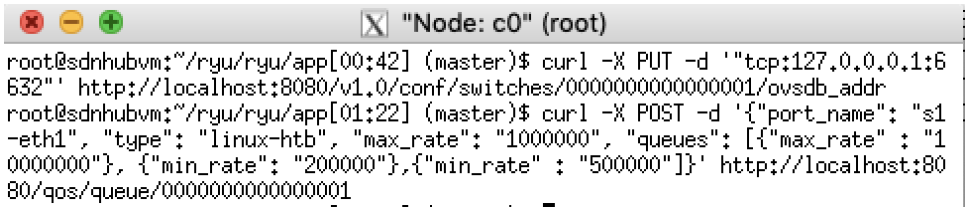


Figure 5.6.3(a) Showing the command to configure Queue settings

- The routers were configured so that they can process packets correctly and flow rules were installed.

Figure 5.6.3(b) Installing Flow Entries on Router 1

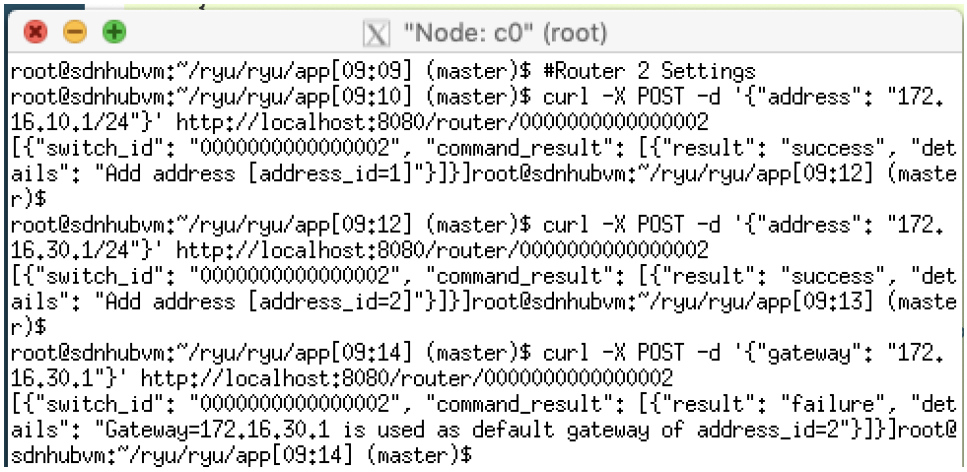


Figure 5.6.3(c) Installing Flow Entries on Router 2

- IP address settings for each router were set, registration of the routers as the default gateway to each host were done as shown in the Figure 5.6.3 (d), (e):

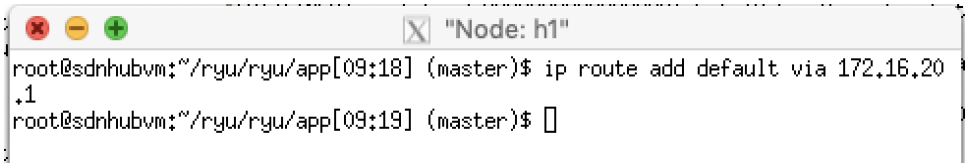


Figure 5.6.3 (d) showing the registration of the gateway route on host h1

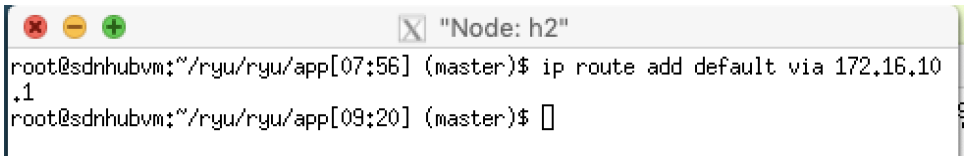


Figure 5.6.3 (e) Showing the registration of the gateway route on host h2

5.6.4 Scalable DiffServ QoS Settings

- Routers were configured with the flow entries in accordance with the priority settings and DSCP values shown in the table below:

Priority	DSCP	Queue ID	QoS ID
1	36(AF31)	1	1
2	34(AF41)	2	2

Table 9: Traffic Classes and DSCP Priority Assignment

- The flow entries installation and DSCP marking were entered in the controller as shown in Figure 5.6.4(a) below:

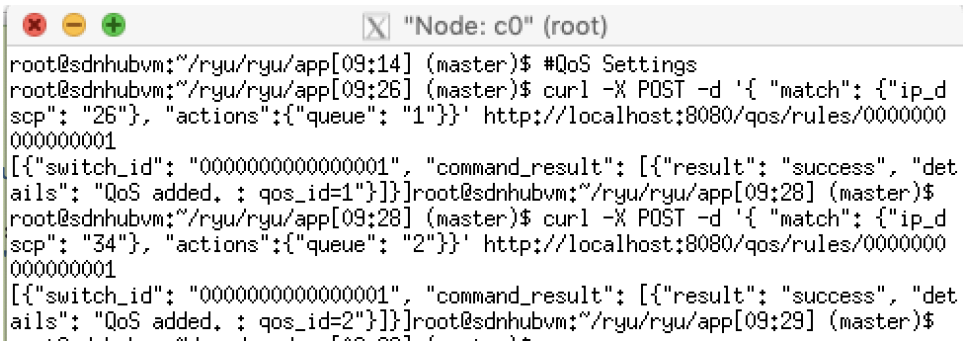


Figure 5.6.4(a) QoS Queue configuration settings.

- The routers were configured with the rules of marking the DSCP values shown in the table:

Priority	Destination Address	Destination Port	Protocol	DSCP	QoS ID
1	172.16.20.10	5002	UDP	26(AF31)	1
1	172.16.20.10	5003	UDP	34(AF41)	2

Table 10: Flow of Prioritized Traffic Settings.

- The routers were configured with the rules of marking the DSCP values as shown in Figure 5.6.4(b):

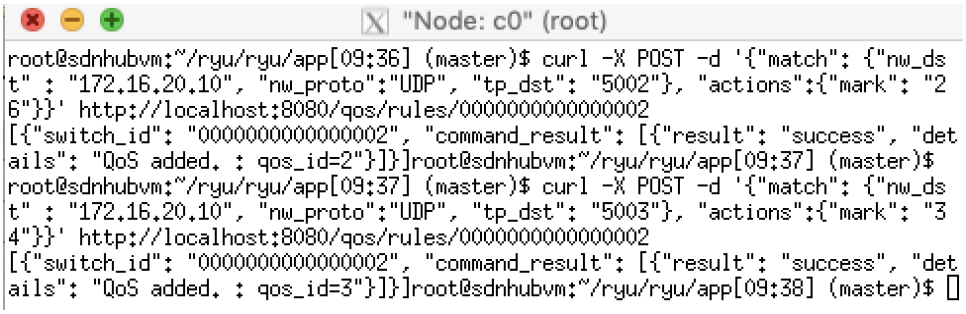


Figure 5.6.4(b) Showing the rules of marking the DSCP values.

- The router configuration settings were verified as shown in Figure 5.6.4(c):

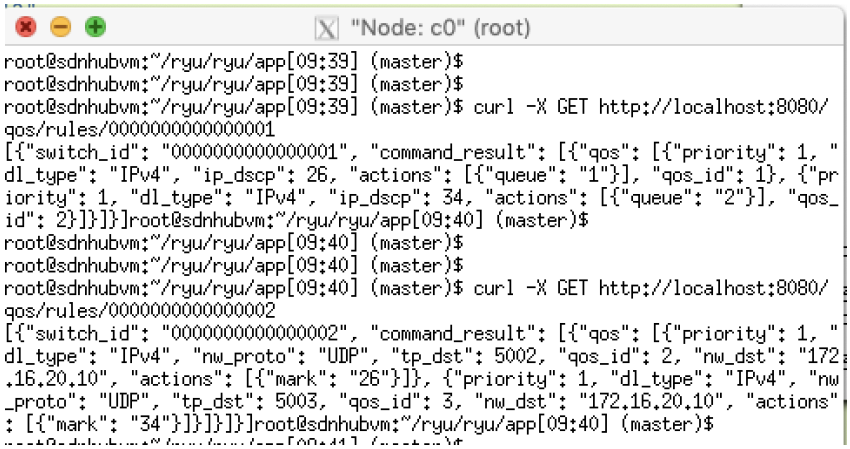


Figure 5.6.4(c) Verifying router settings

5.6.5 Bandwidth Measurement: Scalable DiffServ QoS

Bandwidth measurement was conducted between the client hosts h2 and the server hosts h2 in the following way:

- Server host h1 listens on ports 5001, 5002 and 5003 with UDP protocol with the files saved.
- Client host h2 sends 1Mbps UDP traffic to the port 5001 on h1, sends 300 Kbps UDP traffic to the port 5002 on h1 and sends 600 Kbps UDP traffic on port 5003.
- The server host h1 and client host h2 are shown in Figure 5.6.5(a), (b) below:

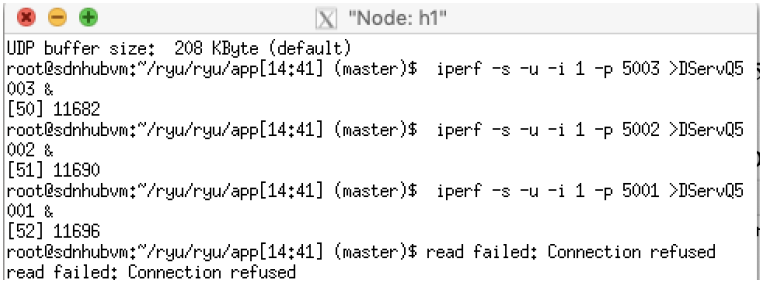


Figure 5.6.5 (a) Server Setup listening on incoming Traffic on UDP Ports 5001, 5002, and 5003.

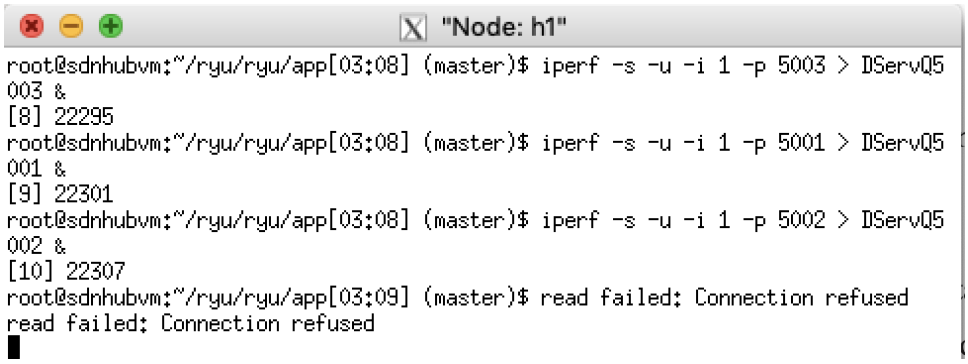


Figure 5.6.5 (b) Client host h2 sending UDP Traffic on Ports 5001, 5002, and 5003.

- The UDP Traffic statistics were saved on the three files and the contents could be retrieved using 'more DServQ5001', 'more DServQ5002', and 'more DServQ5003' commands as shown in Figure 5.6.5(c), (d), (e) below:

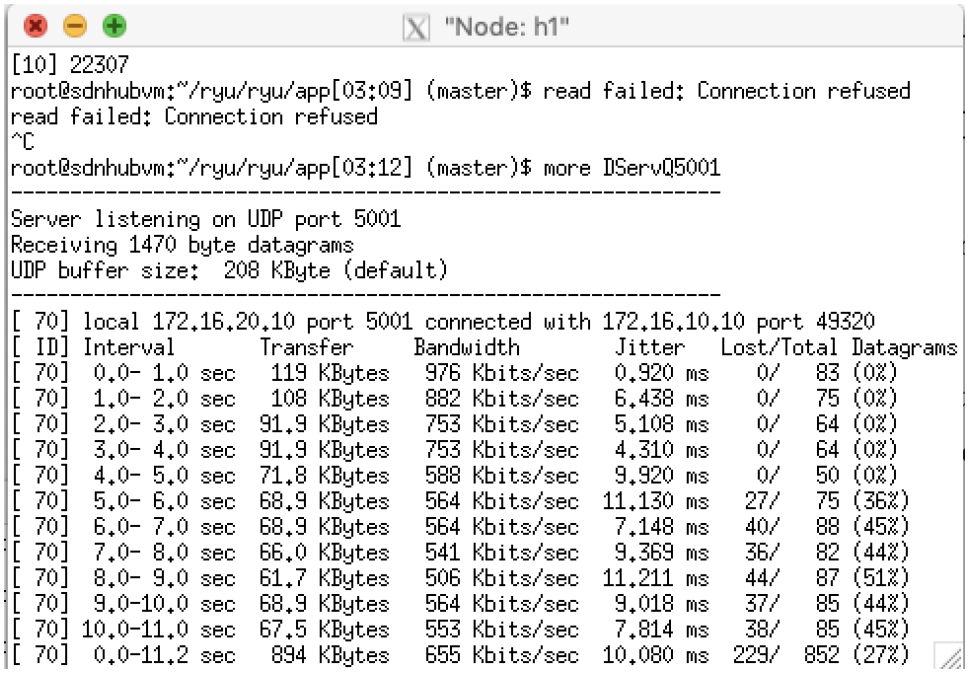


Figure 5.6.5(c) Retrieving the Server Host h1 Port 5001 UDP Traffic file contents.

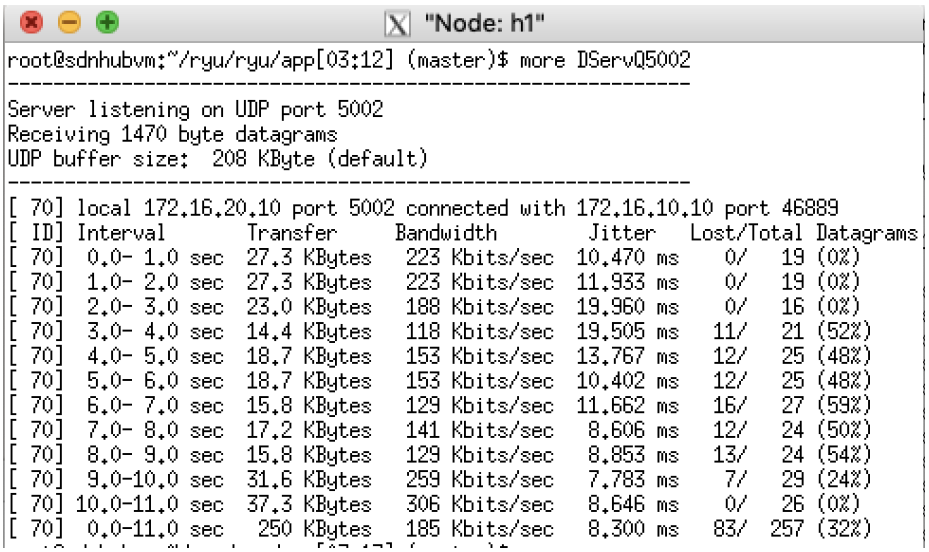


Figure 5.6.5(d) retrieving the Server Host h1 Port 5002 UDP Traffic file contents.

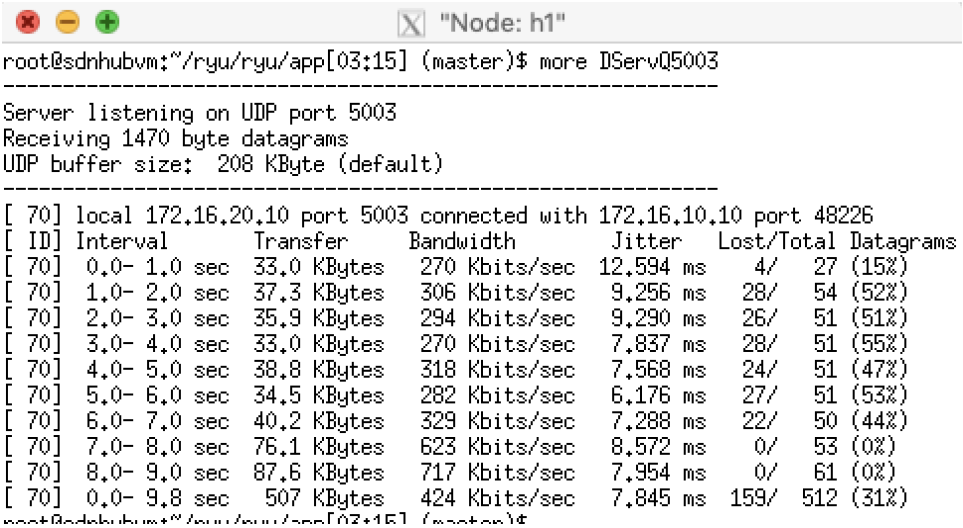


Figure 5.6.5(e) Retrieving the Server Host h1 Port 5003 UDP Traffic file contents.

6. Results and Analysis

6.1 System Model Emulation Tests

6.1.1 Per Flow QoS Operation - Bandwidth Measurement Results

This test was aimed at showing the bandwidth management of the UDP traffic sent to the server host port 5001 which was shaped with up to 600 Kbps and the traffic to the port 5002 is guaranteed 800 Kbps bandwidth. Host h1 the server listens on port 5001 and 5002h UDP protocol. Host h2 (client) sends 1 Mbps UDP traffic to the port 5001 on host h1 (server) and 1 Mbps UDP traffic to port 5002 on h1 (server). The results were obtained in the following way:

- Command ' cat pflow_h1sh2c5001 | grep sec | head –12 | tr - " " | awk '{ print $4, $8} > pflow_h1sh2_band ' was used to extract the time and bandwidth information and save it.
- Gnuplot was used to plot the bandwidth variation with time.
- Figure 6.1.1(a), (b) below shows the commands used and the gnuplot commands used to plot the results.

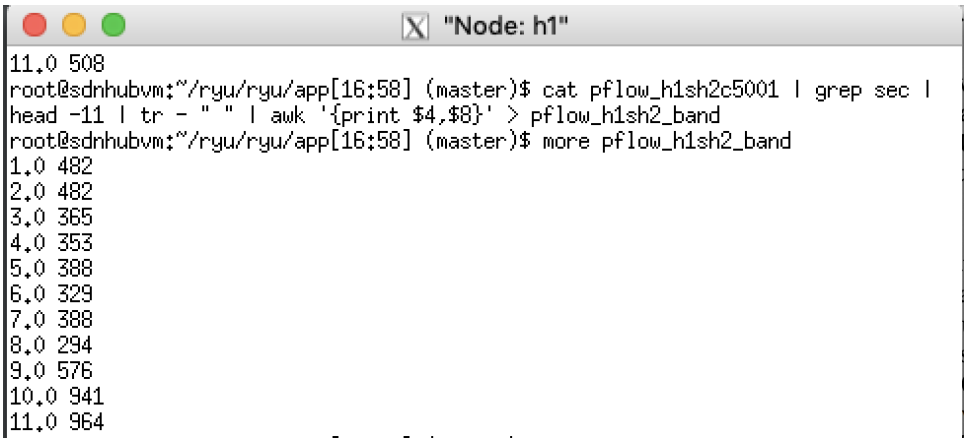


Figure 6.1.1(a) cat command to extract bandwidth time variation and saving it.

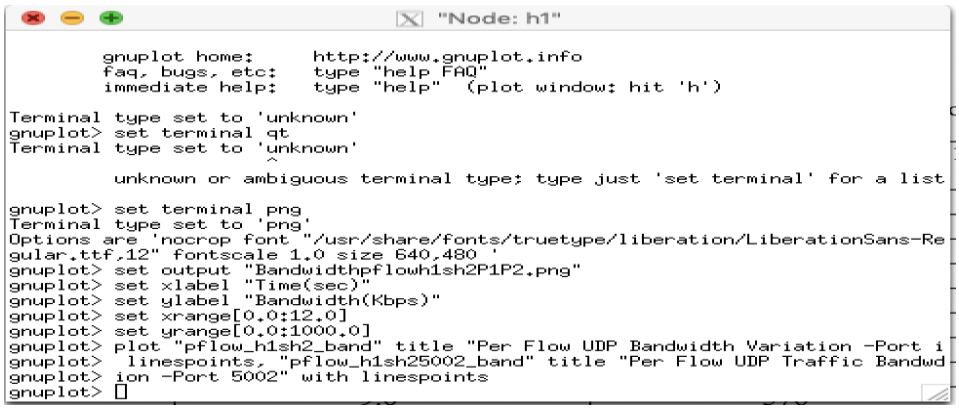


Figure 6.1.1(b) Showing the Gnuplot commands used to plot UDP Traffic Variation.

Time (Sec)	Bandwidth (Kbps) Port 5001	Bandwidth (Kbps) Port 5002
1.0	482	953
2.0	482	964
3.0	365	670
4.0	353	494
5.0	388	529
6.0	329	670
7.0	388	553
8.0	294	635
9.0	576	659
10.0	941	541
11.0	964	670

Table 11: Results UDP Bandwidth Variation with Time

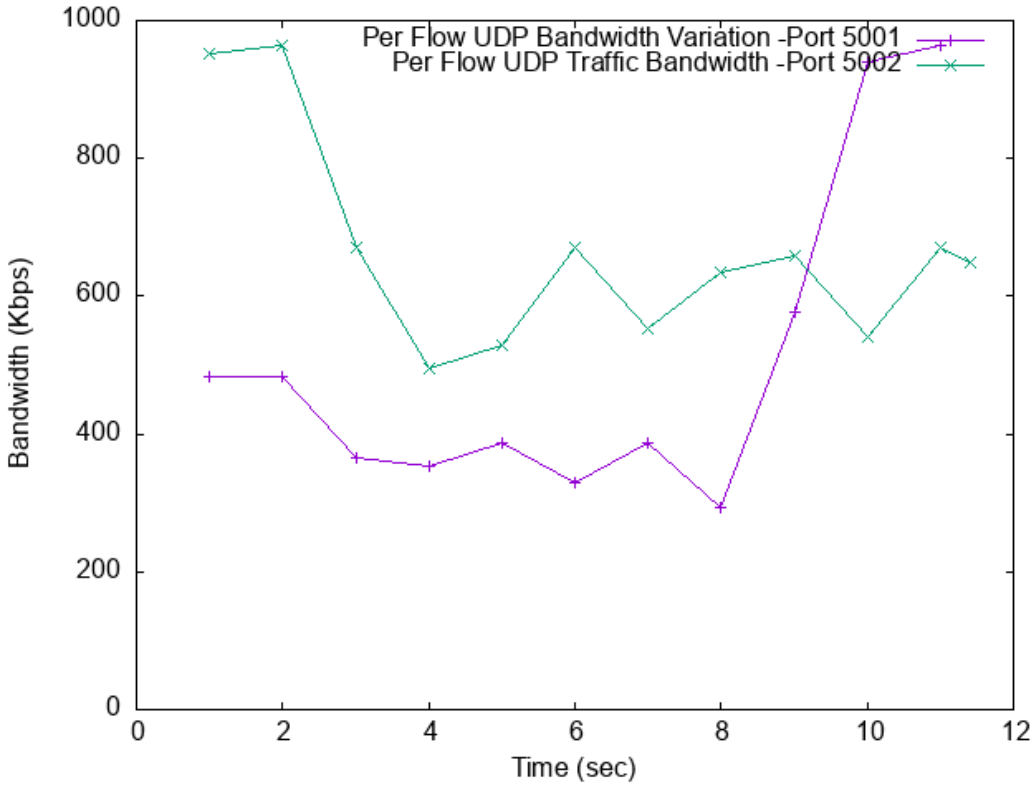


Figure 6.1.1(c) UDP Traffic Variation with Time graph- Port 5001 and Port 5002

6.1.2 Per Flow QoS Operation - Datagram Transfer Rate Results

The datagram transfer rate results for the emulated network scenario are presented below:

Time (Sec)	Transfer Size(KBytes) Port 5001	Transfer Size (KBytes) Port 5002
1.0	58.9	116
2.0	58.9	118
3.0	44.5	81.8
4.0	43.1	60.3
5.0	47.4	64.6
6.0	40.2	81.8
7.0	47.4	67.5
8.0	35.9	77.5
9.0	70.3	80.4
10.0	118	81.8
11.0	685	900

Table 12: Results - UDP Datagram Transfer Sizes with time

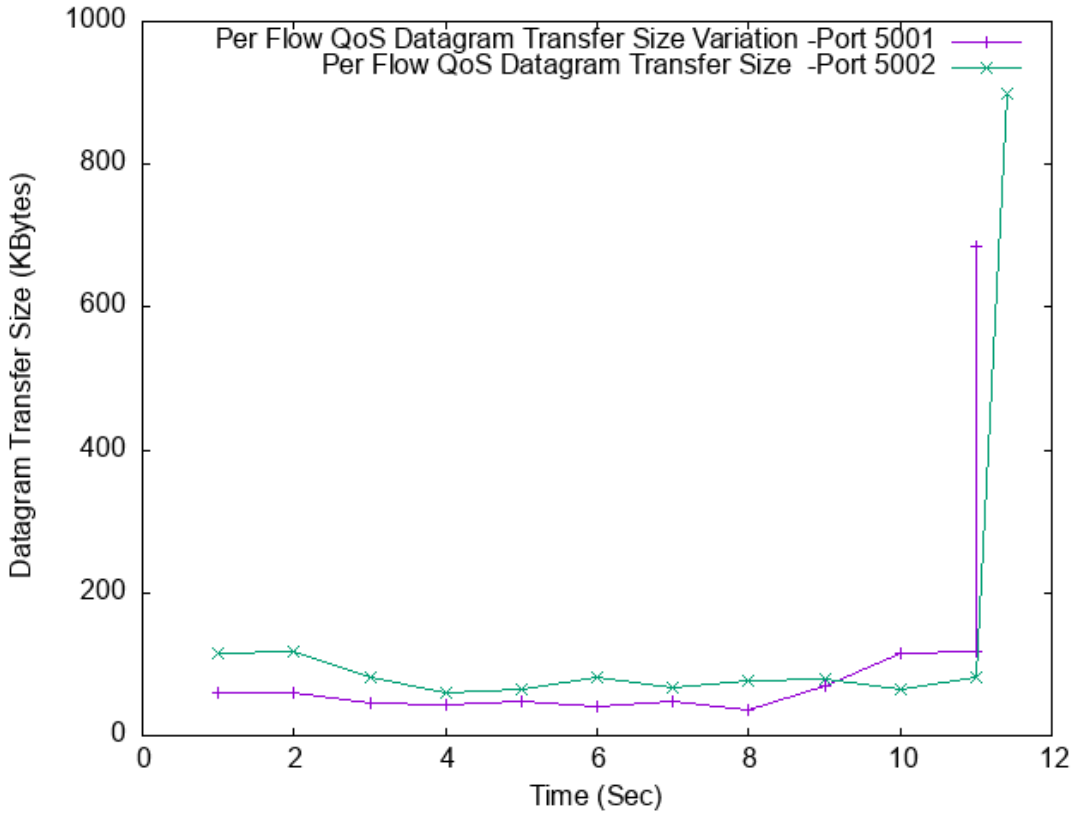


Figure 6.1.2 UDP datagram transfer rate variation on Port 5001 and Port 5002

6.1.3 Per Flow QoS Operation – Jitter Variation Results

The jitter variation for emulated network scenario results are presented below:

Time (Sec)	Jitter (msec) on Port 5001	Jitter (msec) on Port 5002
1.0	12.08	6.208
2.0	12.667	4.784
3.0	8.530	11.146
4.0	7.385	12.454
5.0	7.851	9.524
6.0	9.612	5.347
7.0	8.402	7.224
8.0	6.735	6.493
9.0	7.750	5.988
10.0	5.866	4.693
11.0	6.278	22.288

Table 13: Results -UDP Jitter Variation with Time

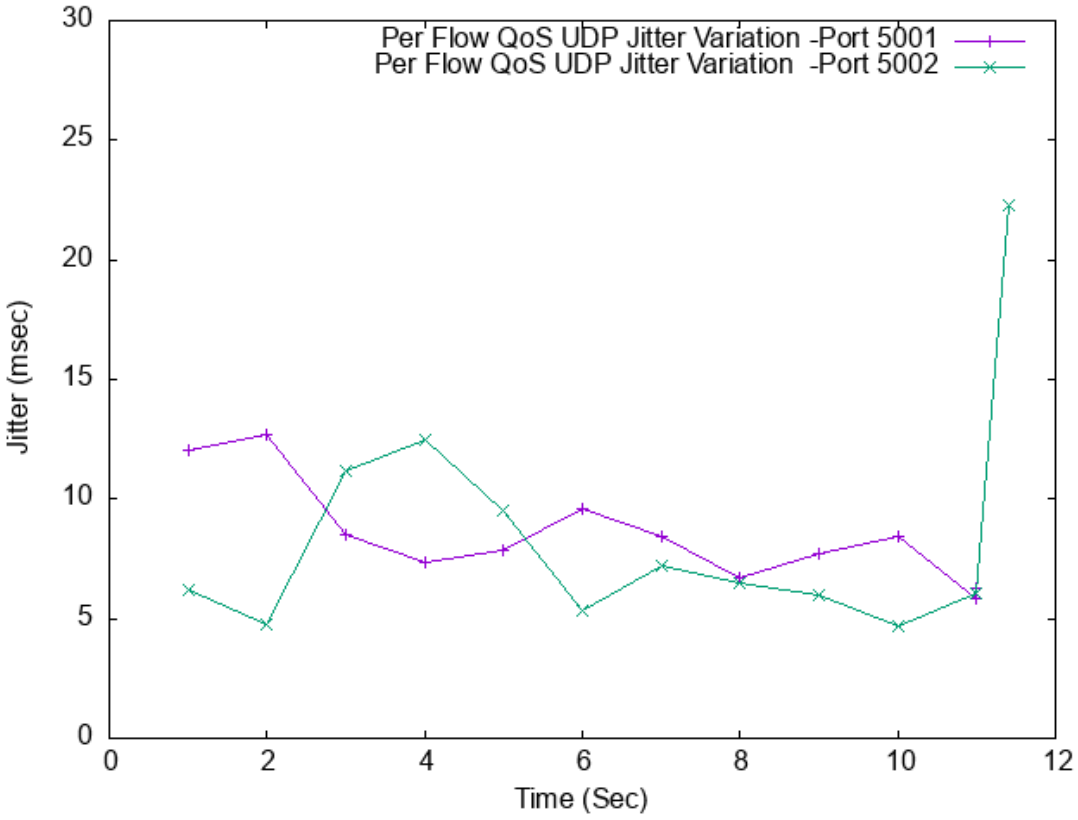


Figure 6.1.3 Showing the UDP Jitter Variation with Time- Port 5001 and Port 5002

6.1.4 Per Flow QoS Operation – Percentage Loss Results

The datagrams that were lost in the emulated network scenario results are presented below:

Time (Sec)	Datagram Percentage Loss (%) on Port 5001	Datagram Percentage Loss (%) on Port 5002
1.0	0.0	0.0
2.0	0.0	0.0
3.0	62.0	0.0
4.0	63.0	33.0
5.0	65.0	33.0
6.0	63.0	45.0
7.0	68.0	37.0
8.0	47.0	33.0
9.0	5.9	45.0
10.0	4.7	33.0
11.0	44	26.0

Table 14: Results -UDP Datagram Percentage Loss Variation with Time.

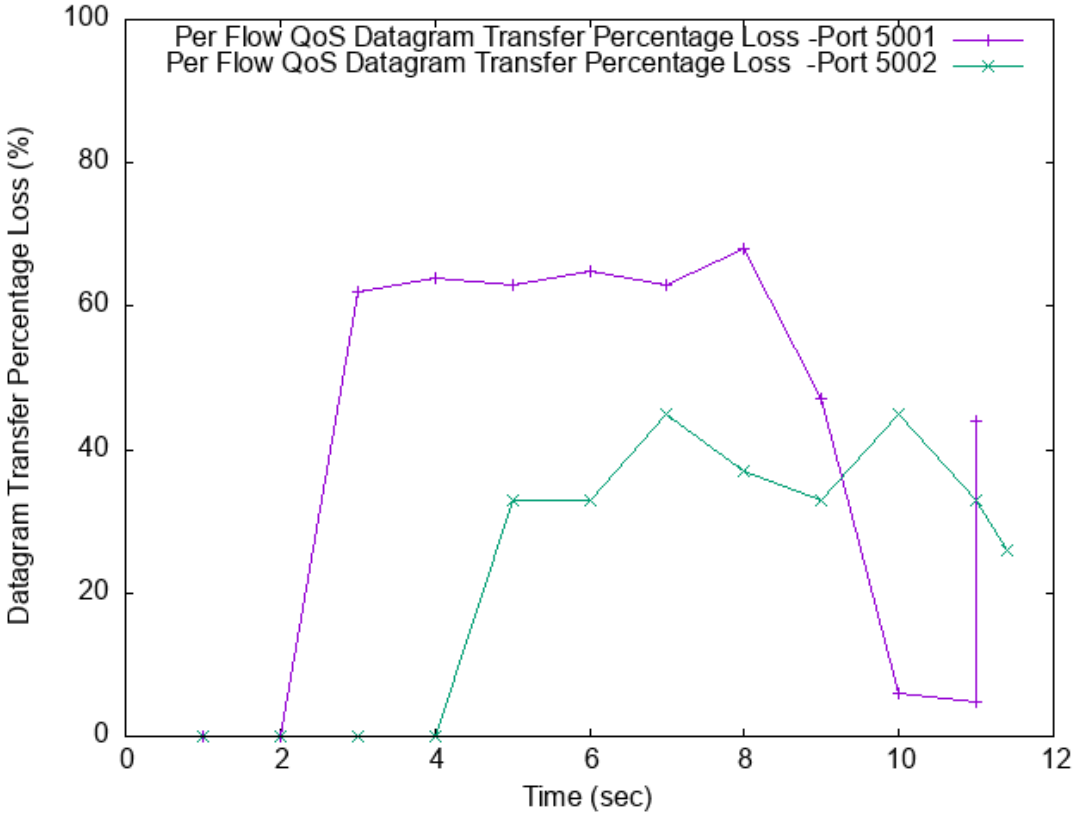


Figure 6.1.4 Showing the UDP Datagram Percentage Loss - Port 5001, Port 5002

Analysis

The results above shows the traffic sent to port 5001 is shaped with up to 500 Kbps and the traffic to the port 5002 is guaranteed with 800 Kbps bandwidth. It can be seen that per flow QoS has the capability to control flows finely but as the communication flows increase, the flow entries which are set for each switch to control the bandwidth also increases. The guaranteed bandwidth traffic sent to 5002 experiences lower jitter levels and lower datagram percentage loss compared to the traffic sent to port 5001 which receives best effort treatment. The traffic guaranteed traffic sent to port 5002 gets better bandwidth allocation and its traffic gets prioritized as depicted in the bandwidth time variation graph and the datagram size transfer rate until when the network experiences congestion which leads to more datagrams being lost and packets being lost as seen towards the end of the packet loss graph.

This clearly shows that while the per-flow QoS can control traffic flows finely, as the communication flows increase, the flow entries which are set for each switch to control the bandwidth also increases. So, the per-flow QoS is not scalable.

6.1.5 Scalable DiffServ QoS - Bandwidth Measurement Results

This test was aimed at showing the bandwidth management of the UDP traffic sent to the server host h1 when UDP Traffic is sent to three ports 5001, 5002 and 5003 in the following way:

- Client host h2 sends 1 Mbps UDP best effort traffic to the server host h1 on port 5001.
- Client host h2 sends 300 Kbps UDP traffic marked AF31 to the server host h1 on port 5002.
- Client host h2 sends 600 Kbps UDP traffic marked AF41 to server host h1 on port 5003.

The results were obtained in the following way:

- The commands ' cat DServQ5001 | grep sec | head –12 | tr - " " | awk '{ print $4, $8} > DServQ5001_band ', ' cat DServQ5002 | grep sec | head –12 | tr - " " | awk '{ print $4, $8} > DServQ5002_band ' and ' cat DServQ5003 | grep sec | head –12 | tr - " " | awk '{ print $4, $8} > DServQ5003_band ' were used to extract the time and bandwidth information and save it.
- Gnuplot was used to plot the bandwidth variation with time.
- The Figure 6.1.5(a), (b) below shows the command used to extract bandwidth information and the gnuplot command used to plot the results.

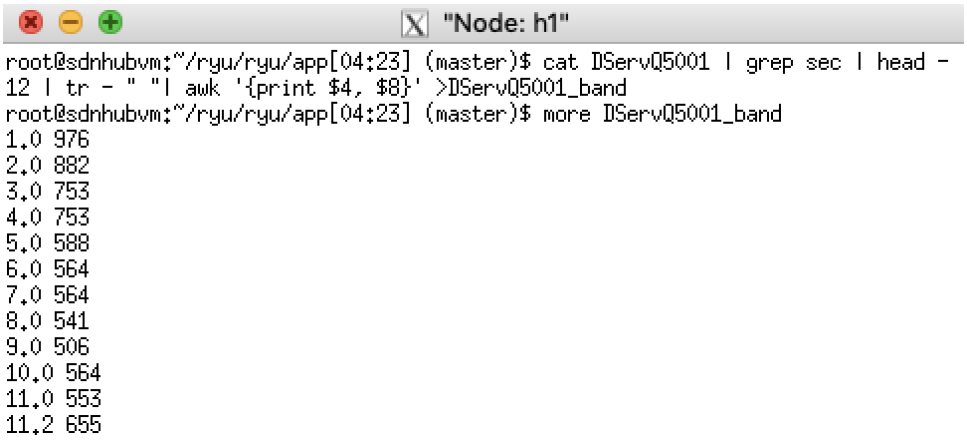


Figure 6.1.5(a) Showing the cat command to extract port 5001 bandwidth time variation

Figure 6.1.5(b) gnuplot commands used to plot UDP Traffic Variation-Port 5002, 5002, 5003

Time (Sec)	Bandwidth (Kbps) Port 5001	Bandwidth (Kbps) Port 5002	Bandwidth (Kbps) Port 5002
1.0	976	223	270
2.0	882	223	306
3.0	753	188	294
4.0	753	118	270
5.0	588	153	318
6.0	564	129	282
7.0	564	141	329
8.0	541	129	623
9.0	506	259	717
10.0	564	306	424
11.0	655	185	308

Table 15: Results - UDP Bandwidth Variation with Time

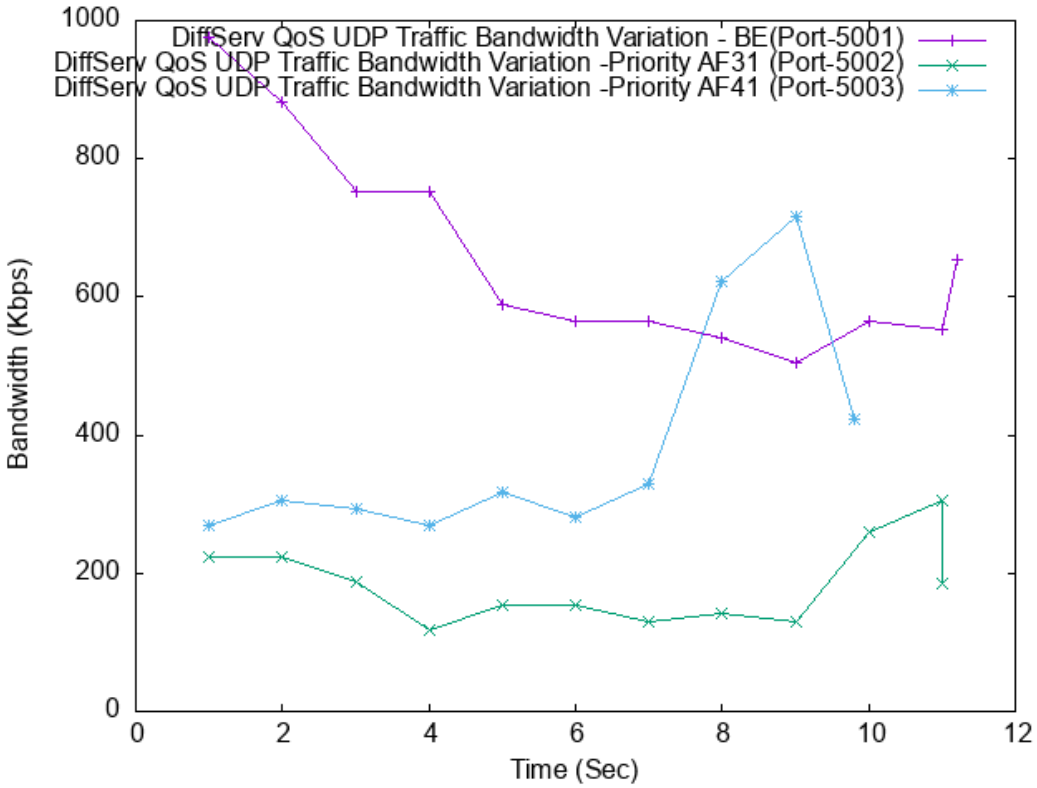


Figure 6.1.5(c) UDP Traffic Variation with Time graph- Port 5002, 5002 and 5003

6.1.6 Scalable DiffServQoS - Datagram Transfer Rate Results

The results are presented below:

Time (Sec)	Transfer Size(KBytes) Port 5001	Transfer Size(KBytes) Port 5002	Transfer Size(KBytes) Port 5003
1.0	119	27.3	33.0
2.0	108	27.3	37.3
3.0	91.9	23.0	35.9
4.0	91.9	14.4	33.0
5.0	71.8	18.7	38.8
6.0	68.9	15.8	34.5
7.0	66.0	17.2	40.2
8.0	61.7	15.8	76.1
9.0	68.9	17.2	80.8
10.0	67.5	31.6	87.6
11.0	894	250	507

Table 16: Results - UDP Datagram Transfer Size Variation with Time

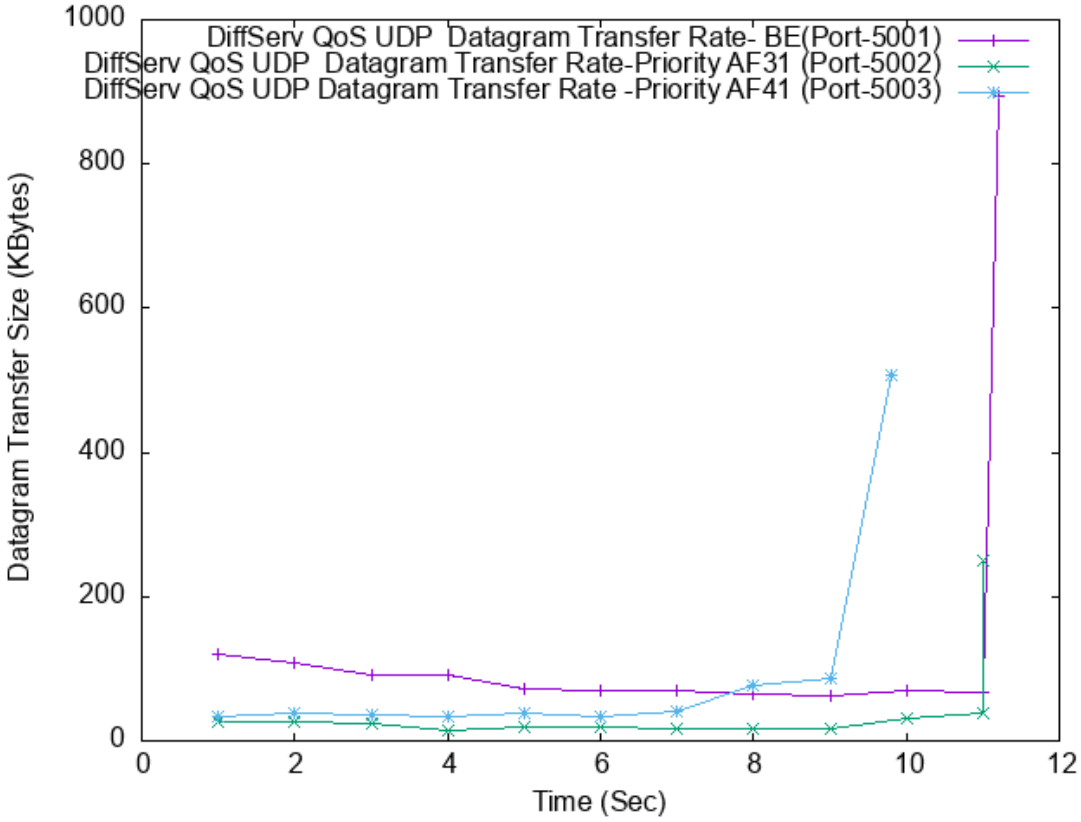


Figure 6.1.6 UDP datagram transfer rate variation on Port 5001, 5002, and 5003.

6.1.7 Scalable DiffServQoS - Jitter Results

This test was aimed at examining the jitter incurred by the UDP traffic sent to the server host h1 traffic is sent to three ports 5001, 5002 and 5003 with traffic marked with priorities and best effort. The results are presented below:

Time (Sec)	Jitter(msec) on Port 5001	Jitter(msec) on Port 5002	Jitter(msec) on Port 5003
1.0	0.920	10.470	12.594
2.0	6.438	11.933	9.256
3.0	5.108	19.960	9.290
4.0	4.310	19.505	7.837
5.0	9.920	13.767	7.568
6.0	11.130	10.402	6.176
7.0	7.148	11.662	7.288
8.0	9.369	8.606	8.572
9.0	11.211	8.853	7.954
10.0	9.018	8.646	7.845
11.0	10.08	8.30	7.80

Table 17: Results - UDP Jitter Variation with Time

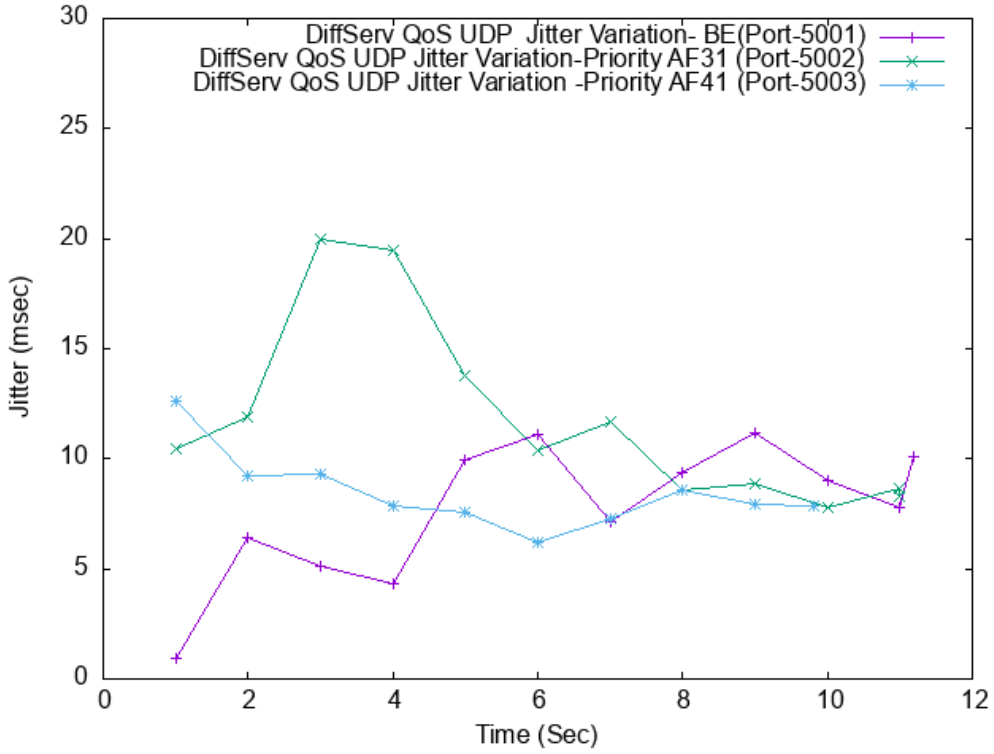


Figure 6.1.7 UDP Jitter Variation with Time- On Ports 5001, 5002, and 5003.

6.1.8 Scalable DiffServQoS - Percentage Loss Results

This test was aimed at examining the percentage loss of the UDP datagram traffic sent to the server host h1 traffic is sent to three ports 5001, 5002 and 5003 in the following way:

- Client host h2 sends 1 Mbps UDP best effort traffic to the server host h1 on port 5001.
- Client host h2 sends 300 Kbps UDP traffic marked AF31 to the server host h1 on port 5002.

- Client host h2 sends 600 Kbps UDP traffic marked AF41 to server host h1 on port 5003.

Time (Sec)	Datagram Percentage Loss (%) on Port 5001	Datagram Percentage Loss (%) on Port 5002	Datagram Percentage Loss (%) on Port 5003
1.0	0	0	15
2.0	0	0	52
3.0	0	0	51
4.0	0	52	55
5.0	0	48	47
6.0	36	48	53
7.0	45	59	44
8.0	44	50	0
9.0	51	54	0
10.0	44	24	31
11.0	45	0	-
11.2	27	32	-

Table 18: Results UDP Datagram Percentage Loss Variation with Time

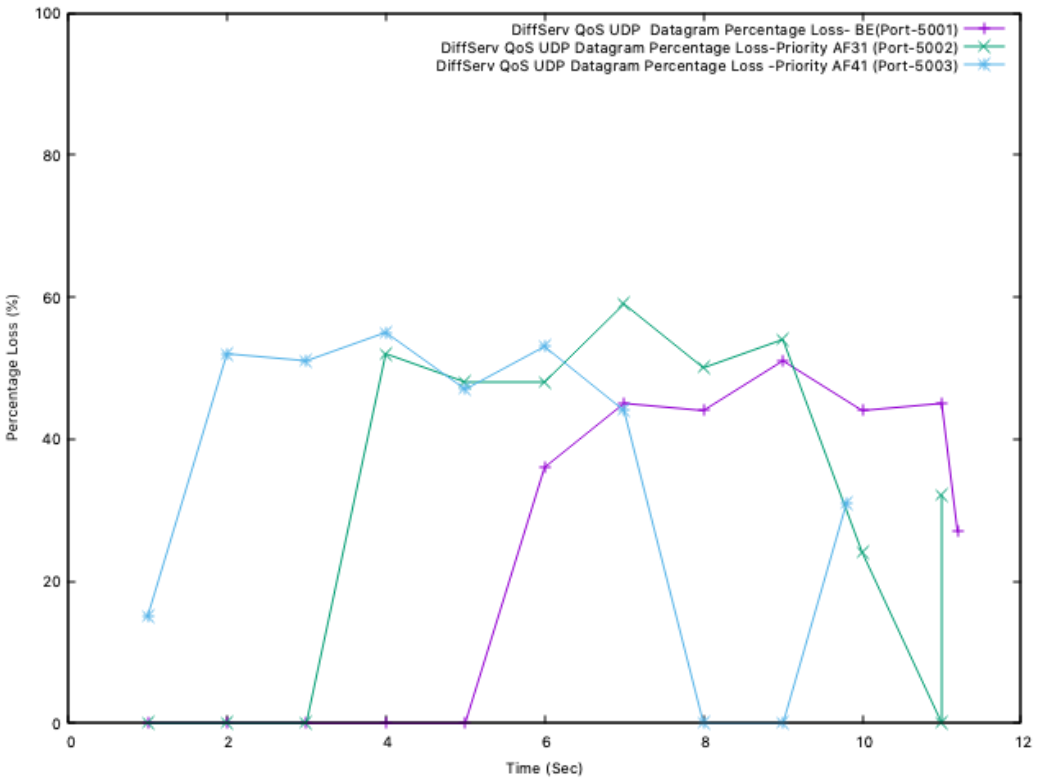


Figure 6.1.8 UDP Datagram Percentage Loss - Port 5001, 5002, and 5003

6.19 Analysis

The above result shows the traffic marked with AF41 which was sent to the port 5003 is guaranteed 500Kbps bandwidth, and the traffic marked with priority AF31 which was sent to the port 5002 is guaranteed 200Kbps bandwidth. On the other hand, the bandwidth of best-effort traffic is limited while the traffic marked with priority AF class is communicating. Traffic with AF41 is prioritized more than traffic with AF 31 which is prioritized more than the BE traffic in terms of bandwidth guarantee allocation and experiences lower jitter levels and lower datagram percentage losses compared to the BE class which receives the worst service levels when the network is congested with all the flows generated by the clients to the server.

To provide scalability, flow aggregation was used which offers a host of benefits:

- The number of flows in the core of networks are reduced, and reduced the complexity associated with per flow management and operations at core routers.
- scheduler efficiency of routers was improved [47]
- When the reserved rate of a flow is coupled with delay as in guaranteed rate schedulers [48], flow aggregation resulted in tighter bounds of the queueing delay.

However, there are known issues with flow aggregation. We outline two known problems: (1) First in First Out (FIFO) aggregation of EF traffic in large arbitrary networks may explode the delay bound after a certain utilization threshold [46] [49] and it is not possible to provide delay bounds for high utilization levels in these networks, and (2) flow aggregation usually needs to be non-work-conserving to be fair to individual constituent flows. Continuous proliferation of very high-capacity links means that the first issue may not be the major problem. The main challenge is the impact of flow aggregation on the resulting QoS provision of the constituent flows of the aggregate.

7. Conclusion

The following conclusions and recommendations are based on the accumulated results of this study.

7.1 Conclusions

The aim of each model and network emulation scenario presented model was to study and prove bandwidth management scalability in SDN. The first model was the general throughput study of the same network under two conditions i.e. 1) Per flow QoS operation and 2) QoS by using DiffServ operation in the SDN environment with ryu controller.

The expectation was to observe better performance of the network when using QoS by DiffServ operation as it the performance scales better and also bandwidth management capabilities are easier implemented through assigning different priorities to the traffic.

The results prove that QoS and bandwidth management in better controlled and monitored when DiffServ is being used rather than applying the per flow QoS approach is being used in the network. QoS by DiffServ operation in the SDN also proved to be more scalable when the network grows compared to per flow QoS which is not scalable.

Per-flow QoS is able to control the network QoS finely but as the communication flows increase, the flow entries which are set for each switch to control the bandwidth also increase, thus proving it difficult for per flow QoS to scale.

The DiffServ QoS divides flows into several QoS classes at the entrance of the DiffServ domain and applies DiffServ to control flows for each class. DiffServ forward the packets according to the PHB defined by the DSCP value that is configured for each class of traffic and realizes QoS.

With the QoS DiffServ offering the capability to mark the traffic, the traffic marked with AF41 (sent to the port 5003) which was set for bandwidth guarantee of 500 Kbps and the traffic marked with AF31 (sent to port 5002) which was set for a bandwidth guarantee of 200 Kbps, the bandwidth of the best effort traffic is limited while the traffic marked with the AF class is communicating. In this way it has been confirmed that it is possible to realize a QoS by using DiffServ model and offers prioritization of traffic which makes bandwidth management easy in the SDN environment.

7.2 Recommendations

The future work which can be done in this topic would be to emulate a network composed of multiple DiffServ domain which will be able to implement traffic metering at the edge of the router and the traffic that exceeds the specified bandwidth to be remarked and treated as low priority class and be able to drop. In the future work one of the things that could be explored would be to use the ToS field in the DSCP value to mark and offer bandwidth guarantee and offer QoS to the different traffic types with defined traffic class priorities. Another recommendation will be to use GNS3 network emulator which offers closer to real network performance monitoring and bandwidth management capabilities.

8. References

[1] Barry, M et al. (2003). "*A brief history of the internet.* Vol. 39. ACM SIGCOMM

[2] Chapter 5 of Roy Fielding's doctoral dissertation introducing REST.

URL: http:// www.ics.uci.edu/~fielding/pubs/dissertation/rest_ arch_style.htm. Date Last Accessed: 25 September 2020.

[3] Floodlight official website: URL: http://www.projectfloodlight.org/ floodlight/. Date Last Accessed: 13 October 2020.

[4] ONF SDN Architecture, Open Networking Foundation, June 2014.

[5] Open vSwitch official website. URL: http://openvswitch.org/. Last Date Accessed: 21 September 2020.

[6] Wang Z. Internet QoS: Architecture and Mechanisms for Quality of Service. The Morgan Kaufmann Series in Networking. Morgan Kaufmann, 1 edition, 03 2001.

[7] Inc. Cisco Systems: Quality of service networking.

URL: http://docwiki.cisco.com/wiki/Quality_of_ Service_Networking. Date Last Accessed: 21 August 2020.

[8] Braden R, Clark D, and Shenker S. Integrated Services in the Internet Architecture: An Overview. ISI and MIT and Xerox PARC, June 1994.

[9] Nichols K, Blake S, Baker F, and Black D. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. Cisco Systems and Torrent Networking Technologies and EMC Corporation, December 1998.

[10] S. Shenker, C. Partridge, and R. Guerin, "Specification of Guaranteed Quality of Service," RFC 2212, Internet Engineering Task Force, September 1997.

[11] J. Wroclawski, “Specification of the Controlled-Load Network Element Service,” RFC 2211, Internet Engineering Task Force, September 1997.

[12] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, “Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification,” RFC 2205, Internet Engineering Task Force, September 1997.

[13] V. Jacobson, K. Nichols, and K. Poduri, “An Expedited Forwarding PHB,” RFC 2598, Internet Engineering Task Force, June 1999, obsoleted by RFC 3246.

[14] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, “Assured Forwarding PHB Group,” RFC 2597, Internet Engineering Task Force, June 1999.

[15] K. Nichols, S. Blake, F. Baker, and D. Black, “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers,” RFC 2474, Internet Engineering Task Force, December 1998.

[16] Configuration Guide - QoS. HUAWEI TECHNOLOGIES Company LTD, March 2012.

[17] Nichols K, Blake S, Baker F, and Black D. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. Cisco Systems and Torrent Networking Technologies and EMC Corporation, December 1998.

[18] Ramakrishnan K, Floyd S, and Black D. The Addition of Explicit Congestion Notification (ECN) to IP. TeraOptic Networks and ACIRI and EMC, September 2001

[19] Grossman D. New Terminology and Clarifications for Diffserv. Motorola, Inc., 2002.

[20] Cisco Nexus 1000V Quality of Service Configuration Guide. Cisco, release 4.2(1) sv1 (4) edition, 03 2016.

[21] Scheduling and policing mechanisms, June 2019. URL http:// netlab.ulusofona.pt/rc/book/6-multimedia/6_06/index.htm. Date Last Accessed: 13 October 2020

[22] Network OS Layer 2 Switching Configuration Guide, 6.0.1a. Brocade Communications Systems, Inc., 53-1003770-06 edition, 08 2016.

[23] Cisco IOS Quality of Service Solutions Configuration Guide. Cisco release 12.2 edition, January 2014.

[24] OpenFlow Switch Specification. Open Networking Foundation, version 1.3.0 edition, June 2012.

[25]Mininet Team. Getting Started with Mininet, 05 2019. URL http://mininet.org/download/. Date Last Accessed: 12 October 2020

[26] VMware. Vmware learning zone, 08 2017. URL: https: //www.vmware.com/professional-services/learning-zone. html?src=WWW_US_HP_LearningZone_R2C2_D_NA_StartTrial. Date Last Accessed: 14 October 2020

[27] National Library of the Netherlands. What is emulation? 05 2019. URL https://www. kb.nl/en/organisation/research-expertise/research-on-digitisation-and-digital-preservation/ emulation/what-is-emulation. Date Last Accessed: 15 October 2020

[28] Linux Foundation Collaborative Project. Quality of service (qos), 05 2019. URL: http://docs.openvswitch.org/en/latest/faq/ qos/. Date Last Accessed: 16 October 2020

[29] OpenFlow1.3software switch, 05 2017. URL https://github. Com/CPqD/ofsoftswitch13. Date Last Accessed: 17 October 2020.

[30]Openflow1.3 Tutorial. URL https://github.com/CPqD/ ofsoftswitch13/wiki/OpenFlow-1.3-Tutorial. Date Last Accessed: 18 October 2020

[31] Khondoker R, Zaalouk A, Marx R, and Bayarou K. Feature-based comparison and selection of software defined networking (SDN) controllers. Computer Applications and Information Systems (WCCAIS), 2014 World Congress, January 2014.

[32] Zimarina D Pashkov V Smeliansky R Shalimov A, Zuikov D. Advanced study of sdn/openflow controllers. Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, (1), October 2013.

[33] Jerez Chaves L, Calciolari Garcia I, and Madeira E. Ofswitch13: Enhancing ns-3 with openflow 1.3 support. WNS3 '16, Proceedings of the Workshop on ns-3, pages 33–40, 06 2016.

[34] A Comprehensive Survey of Interface Protocols for Software Defined Networks. URL: https://arxiv.org/pdf/1902.07913.pdf . Date Last Accessed: 23 October 2020

[35] SDN vs NFV Understanding Their Differences, Similarities and Benefits. URL: https://blog.equinix.com/blog/2020/03/10/sdn-vs-nfv-understanding-their-differences-similarities-and-benefits/ . Date Last Accessed : 26 October 2020

[36] Li G, Dong M, Ota K, et al. Deep Packet Inspection Based Application-Aware Traffic Control for Software Defined Networks. Global Communications Conference. IEEE, 2017:1-6.

[37] Jeong S, Lee D, Choi J, et al. Application-aware Traffic Management for OpenFlow networks. Network Operations and Management Symposium. IEEE, 2016:1-5.

[38] Qazi Z A, Lee J, Jin T, et al. Application-awareness in SDN. Computer Communication Review, 2013, 43(4):487-488

[39] Roughan M, Sen S, Spatscheck O, et al. Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification. ACM SIGCOMM Conference on Internet Measurement. ACM, 2004:135-148

[40] Moore A W, Zuev D. Internet traffic classification using Bayesian analysis techniques. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. ACM, 2005:50-60

[41] Nguyen, Thuy TT, and Grenville J. Armitage. “Training on multiple subflows to optimise the use of Machine Learning classifiers in real-world IP networks.” LCN. 2006.

[42] Reza M, Javad M, Raouf S, et al. Network Traffic Classification using Machine Learning Techniques over Software Defined Networks. International Journal of Advanced Computer Science & Applications, 2017, 8(7)

[43] McGregor, Anthony, et al. “Flow clustering using machine learning techniques.” Passive and Active Network Measurement. Springer Berlin Heidelberg, 2004. 205-214

[44] Erman, Jeffrey, Martin Arlitt, and Anirban Mahanti. “Traffic classification using clustering algorithms.” Proceedings of the 2006 SIGCOMM workshop on Mining network data. ACM, 2006

[45] Li Y, Li J. MultiClassifier: A combination of DPI and ML for application-layer classification in SDN. International Conference on Systems and Informatics. IEEE, 2015:682-686.

[46] Shi J, Chung S-H. A traffic-aware quality-of-service control mechanism for software-defined networking based virtualized networks. *International Journal of Distributed Sensor Networks*. March 2017. Doi: 10.1177/1550147717697984.

[46] Charny, A., Le Boudec, J.-Y.: Delay bounds in a network with aggregate scheduling. In: Proceedings of Quality of Future Internet Services, pp. 1–13 (2000) [47] Cobb, J.A.: Preserving quality of service guarantees in spite of flow aggregation. IEEE/ACM Trans. on Networking 10(1), 43–53 (2002)

[48] Goyal, P., Lam, S.S., Vin, H.M.: Determining end-to-end delay bounds in heterogeneous networks. Multimedia System 5(3), 157–163 (1997)

[49] Jiang, Y.: Delay bounds for a network of guaranteed rate servers with fifo aggregation. Computer Networks 40(6), 683–694 (2002)

[50] F. Shahrokhi and D. Matula. The maximum concurrent flow problem. Journal of the ACM, 37(2):318–334, 1990.

9. Appendices

9.1. Appendix A Mininet command lines screen shots and results

```
ubuntu@sdnhubvm:~[11:11]$ sudo mn --mac --switch ovsk --controller remote -x
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Running terms on localhost:10.0
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Figure A. 1 Mininet command for linear topology creation

9.2 Controller Configuration and Settings Verification

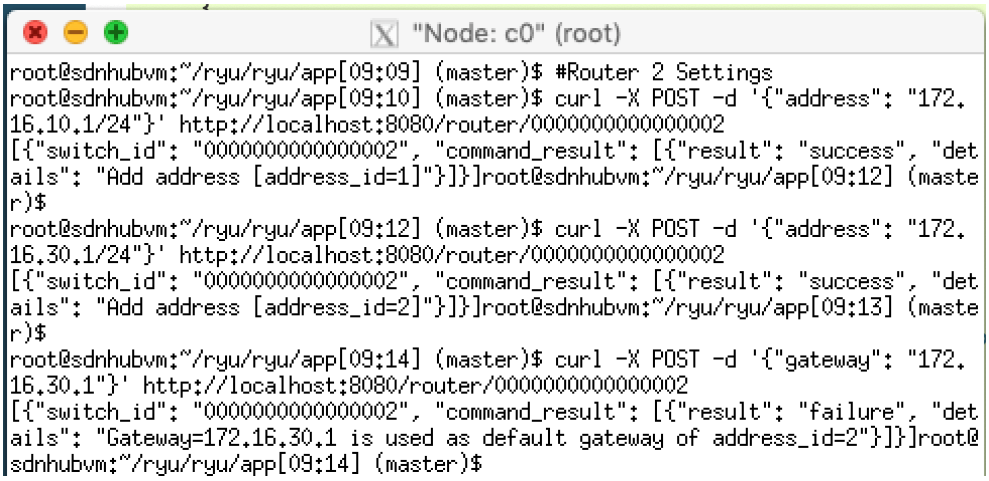


Figure A. 2 Setting Router QoS Configuration

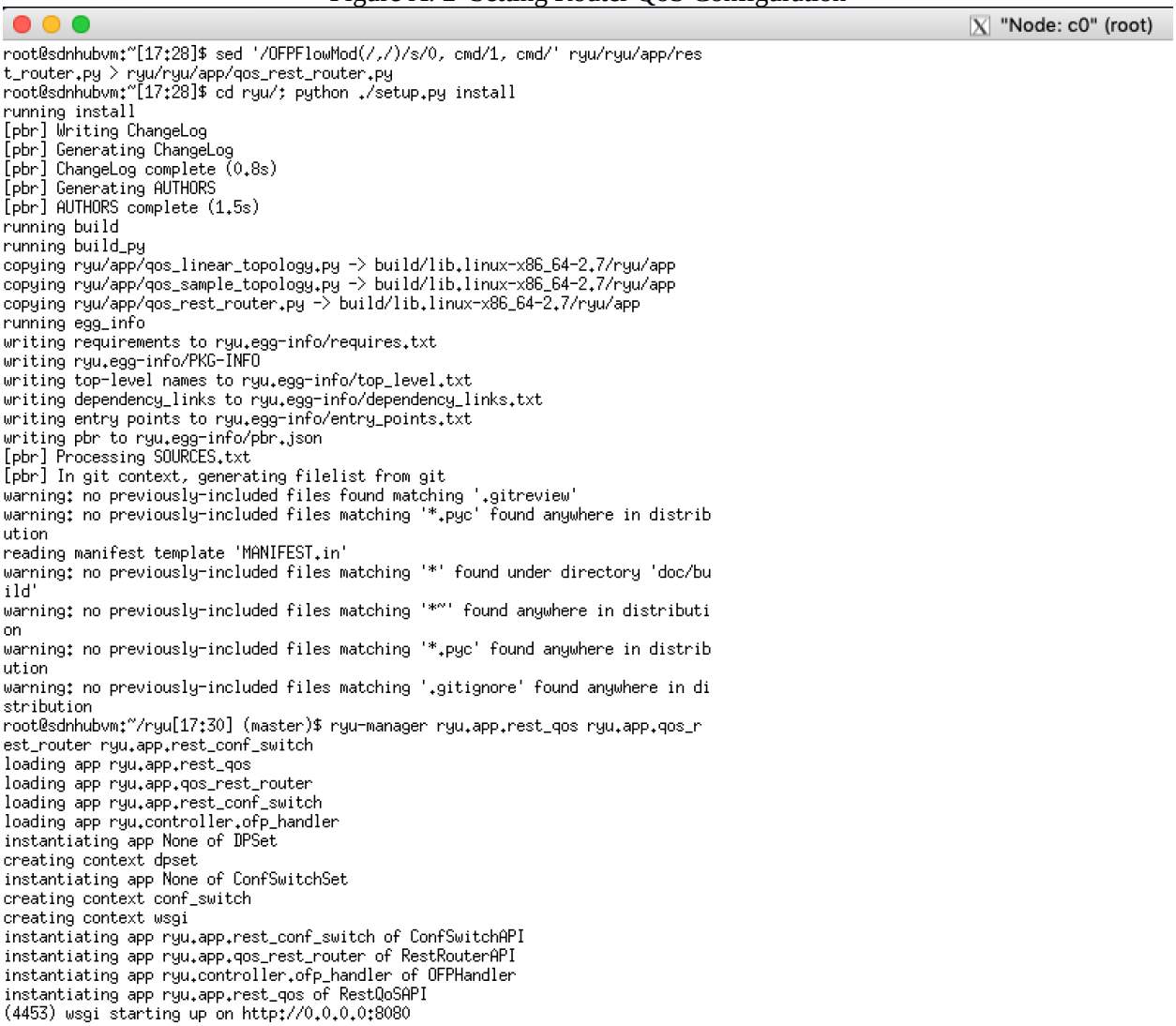


Figure A. 3 SDN QoS Controller Settings

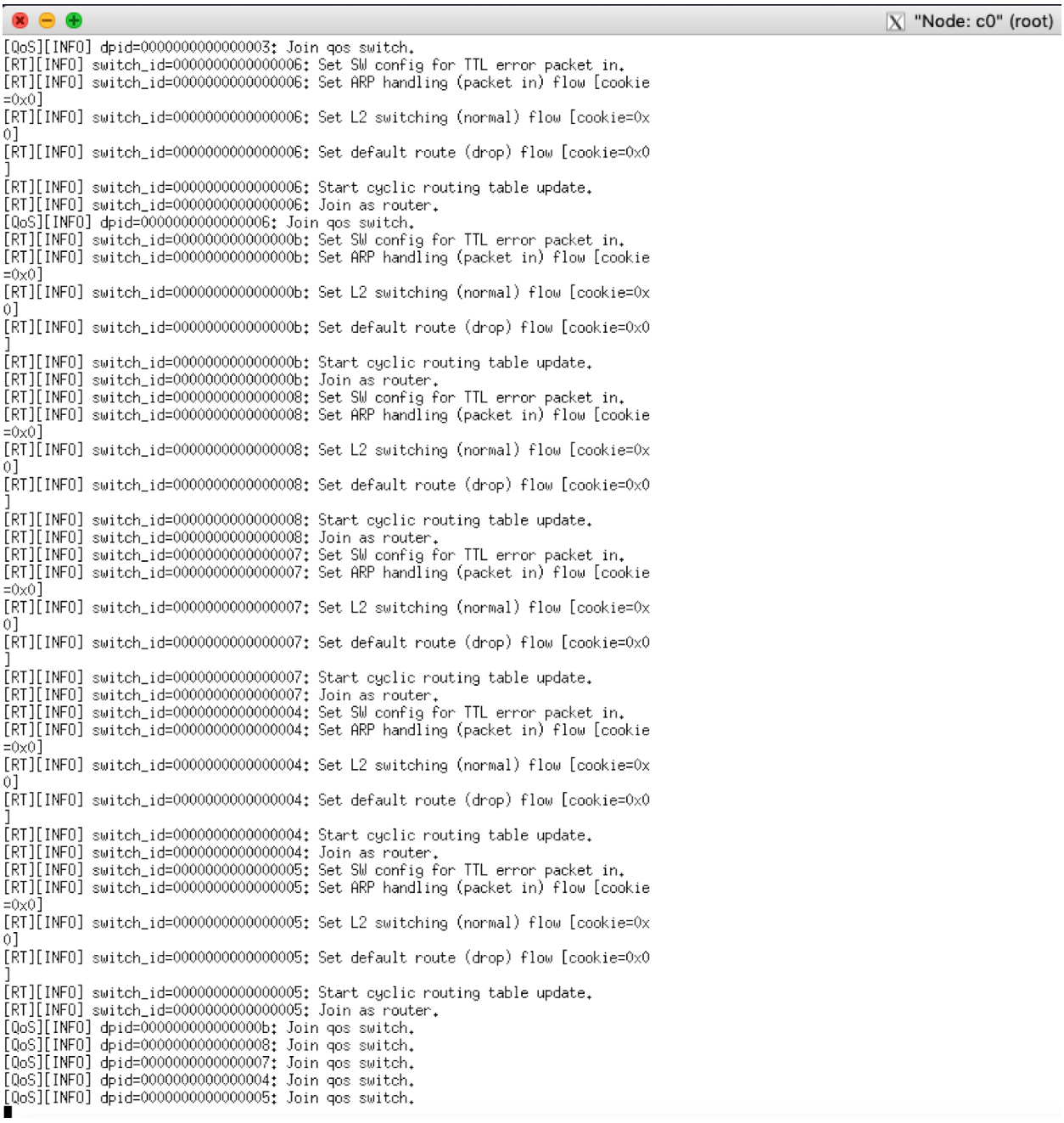


Figure A. 4 SDN QoS Controller Settings

9.3 Controller QoS Configuration Settings

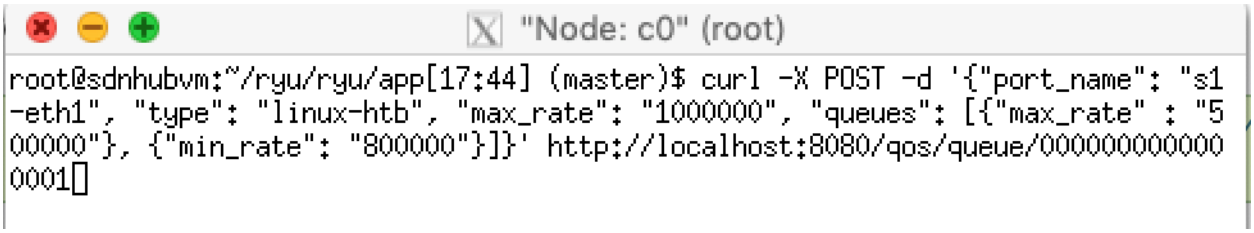


Figure A. 5 Installing Flow Rules : QoS Controller Settings

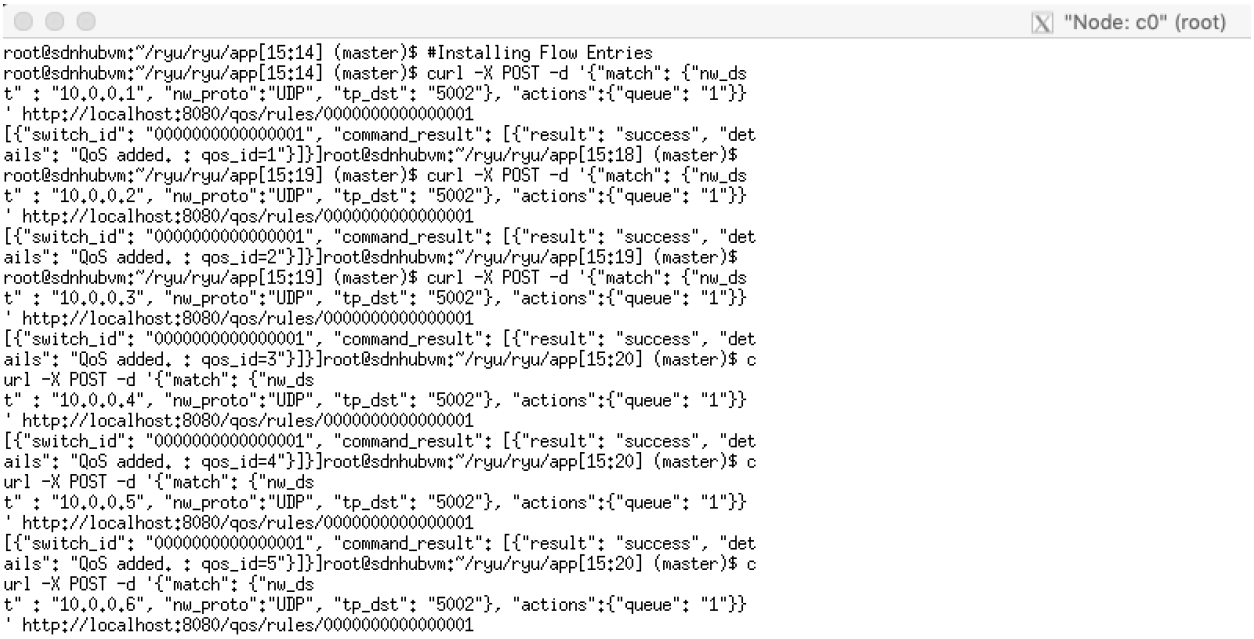


Figure A. 6 Installing Flow Rules: QoS Controller Settings

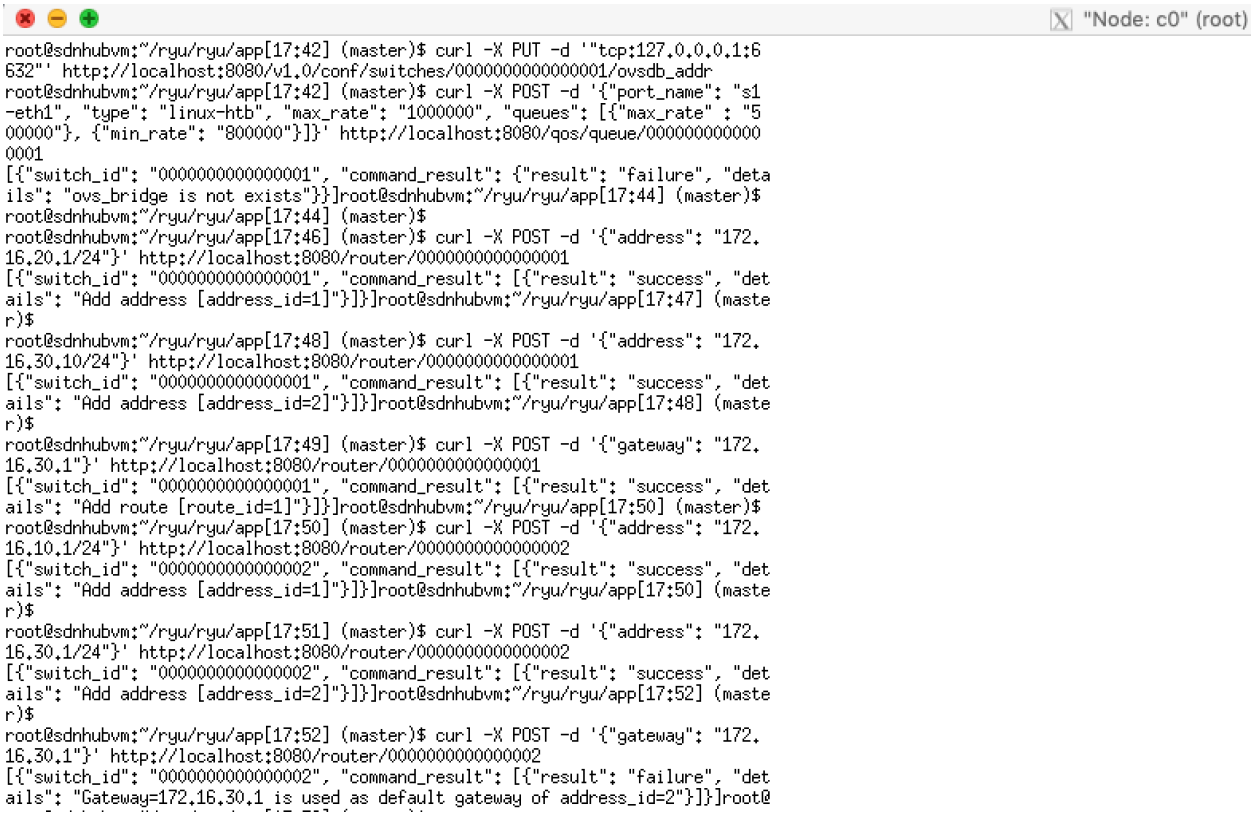


Figure A. 7 Installing Flow Rules: QoS Controller Settings

9.4 Verification of Traffic Class Priorities and DSCP Settings

Figure A. 8 Verifying the Flow Rules Installation in the SDN Controller

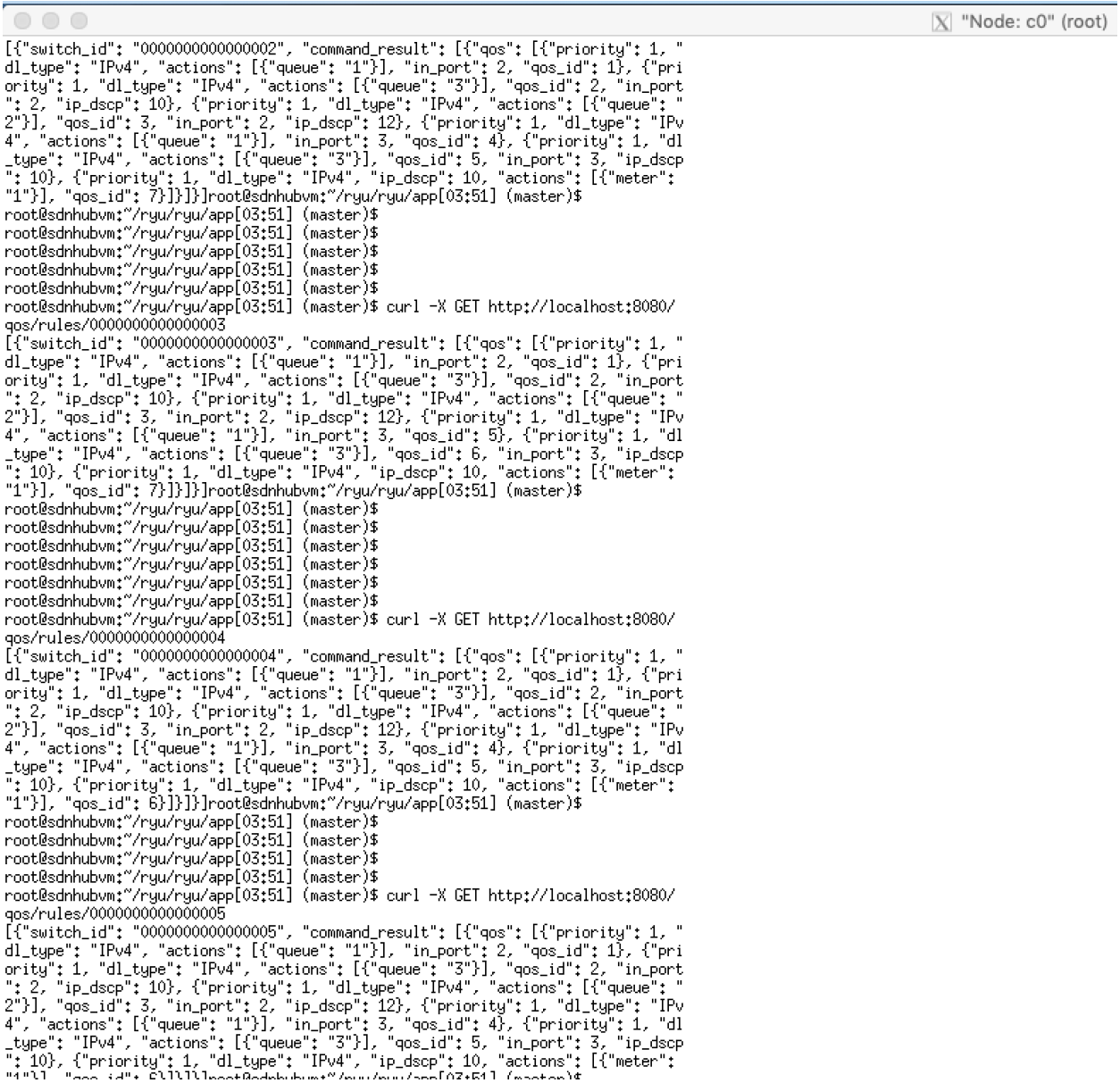


Figure A. 9 Verifying the Flow Rules Installation in the SDN Controller

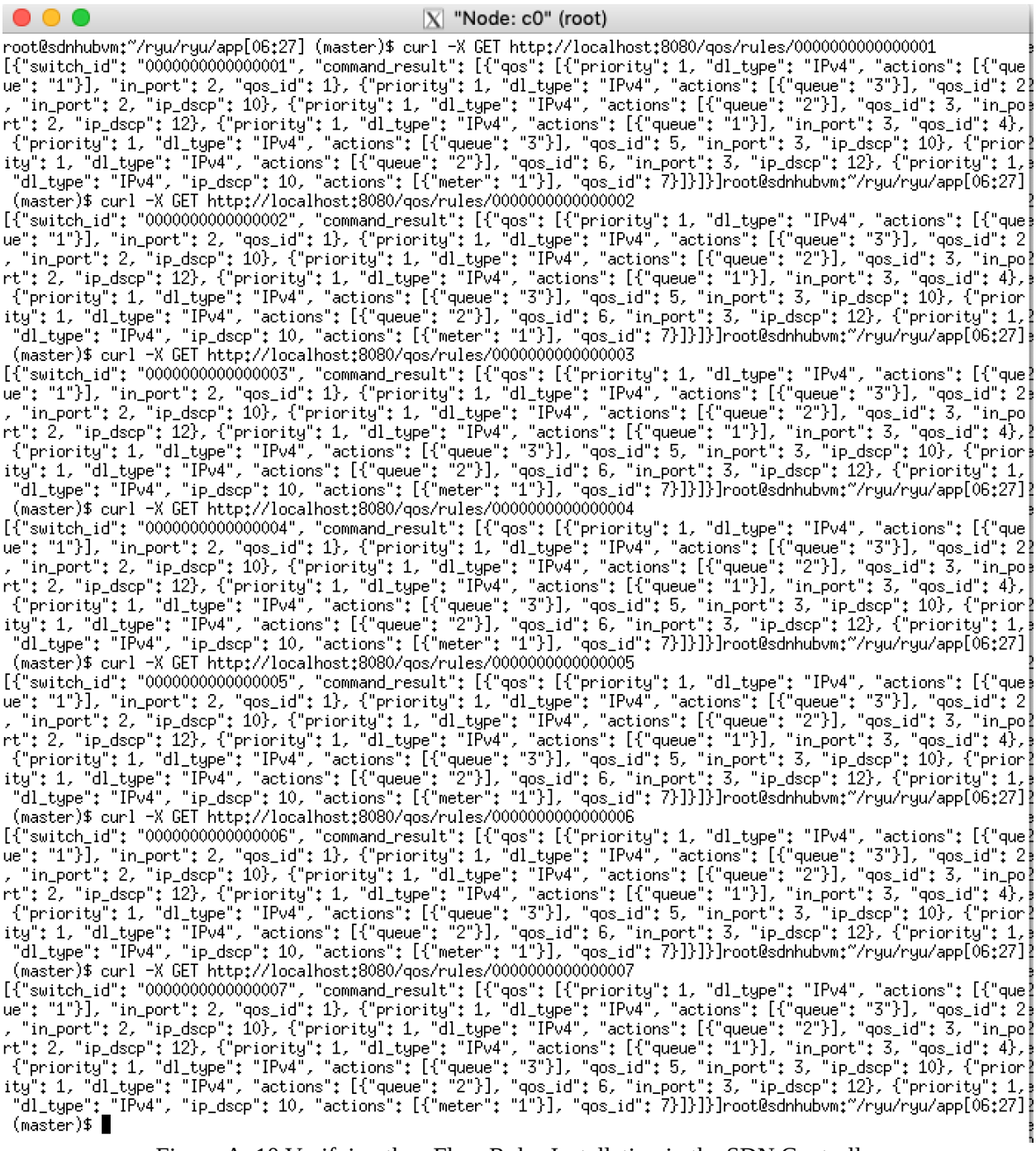


Figure A. 10 Verifying the Flow Rules Installation in the SDN Controller

9.5 Switch Settings

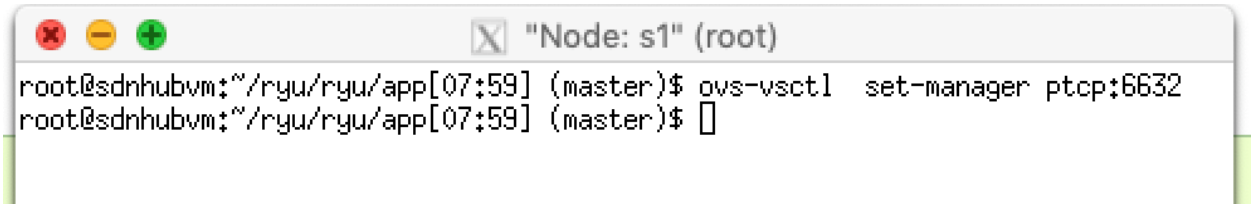


Figure A. 11 Switch Configurations: Connecting with SDN Controller

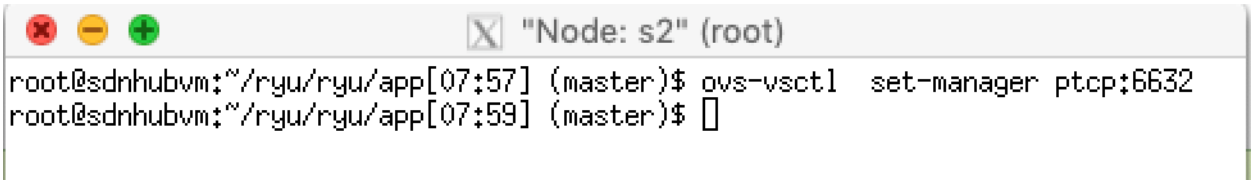


Figure A. 12 Switch Configurations: Connecting with SDN Controller

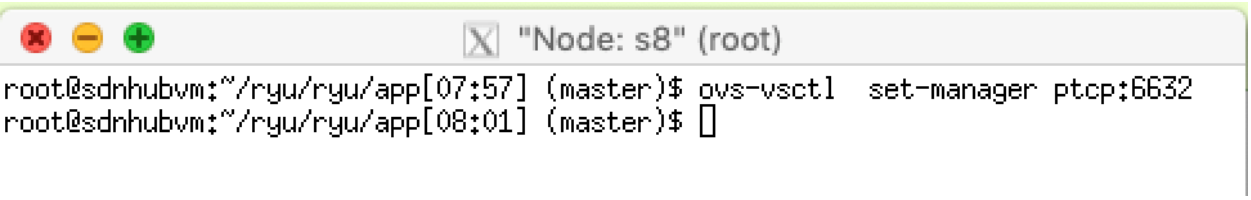


Figure A. 13 Switch Configurations: Connecting with SDN Controller

9.6 Network Nodes IP Settings

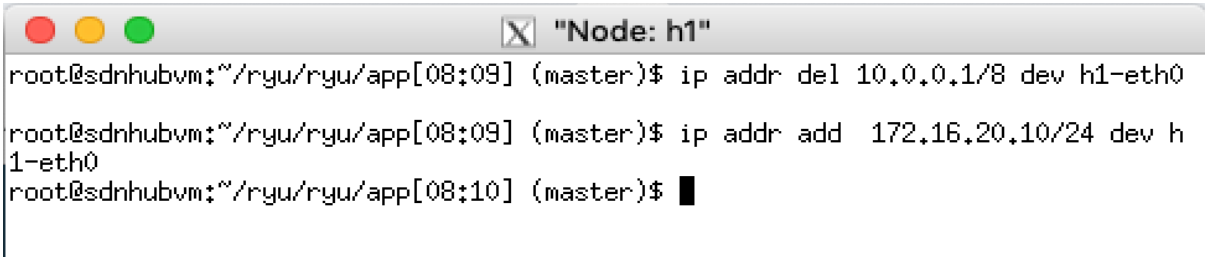


Figure A. 14 Network Node IP Address Settings Configuration

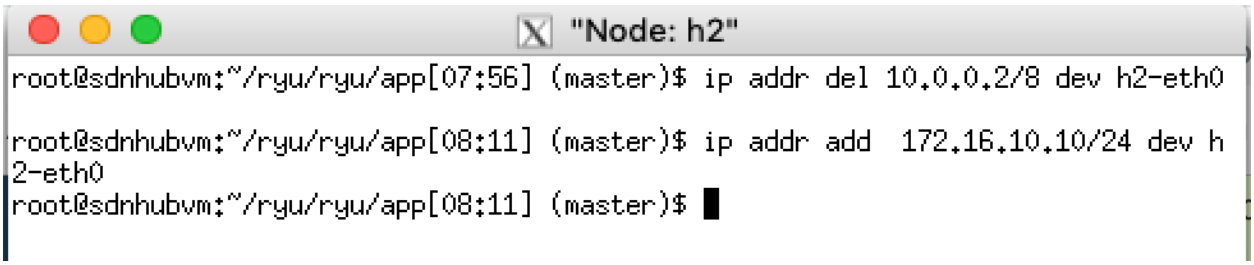


Figure A. 15 Network Node IP Address Settings Configuration

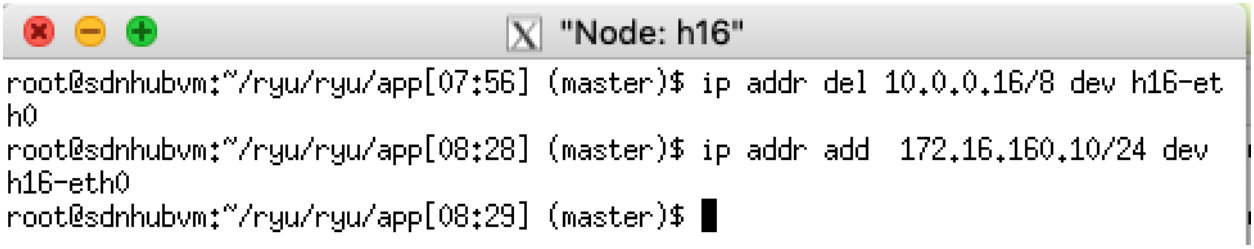


Figure A. 16 Network Node IP Address Settings Configuration

9.7 Client Server Settings

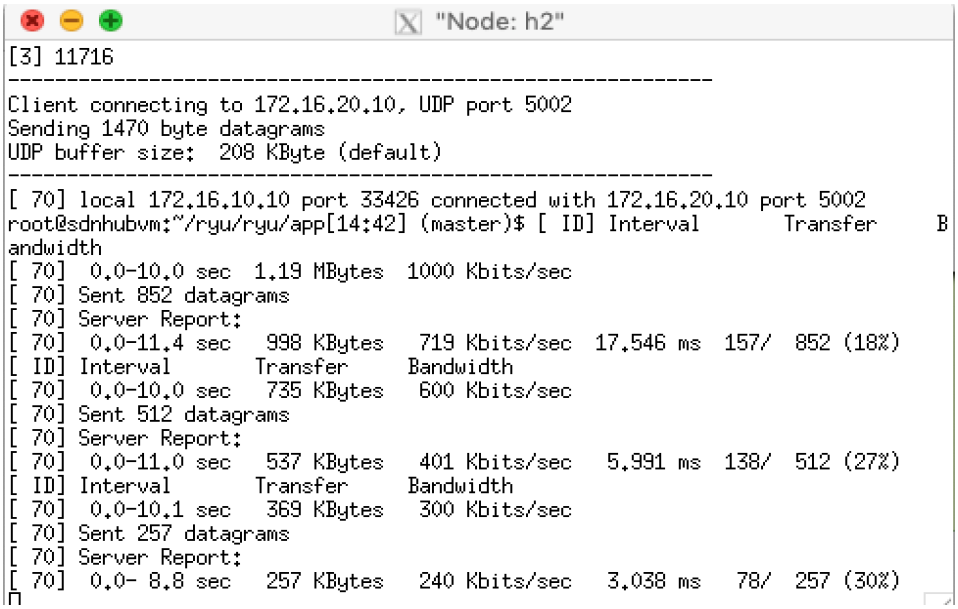


Figure A. 17 Client Server Connection

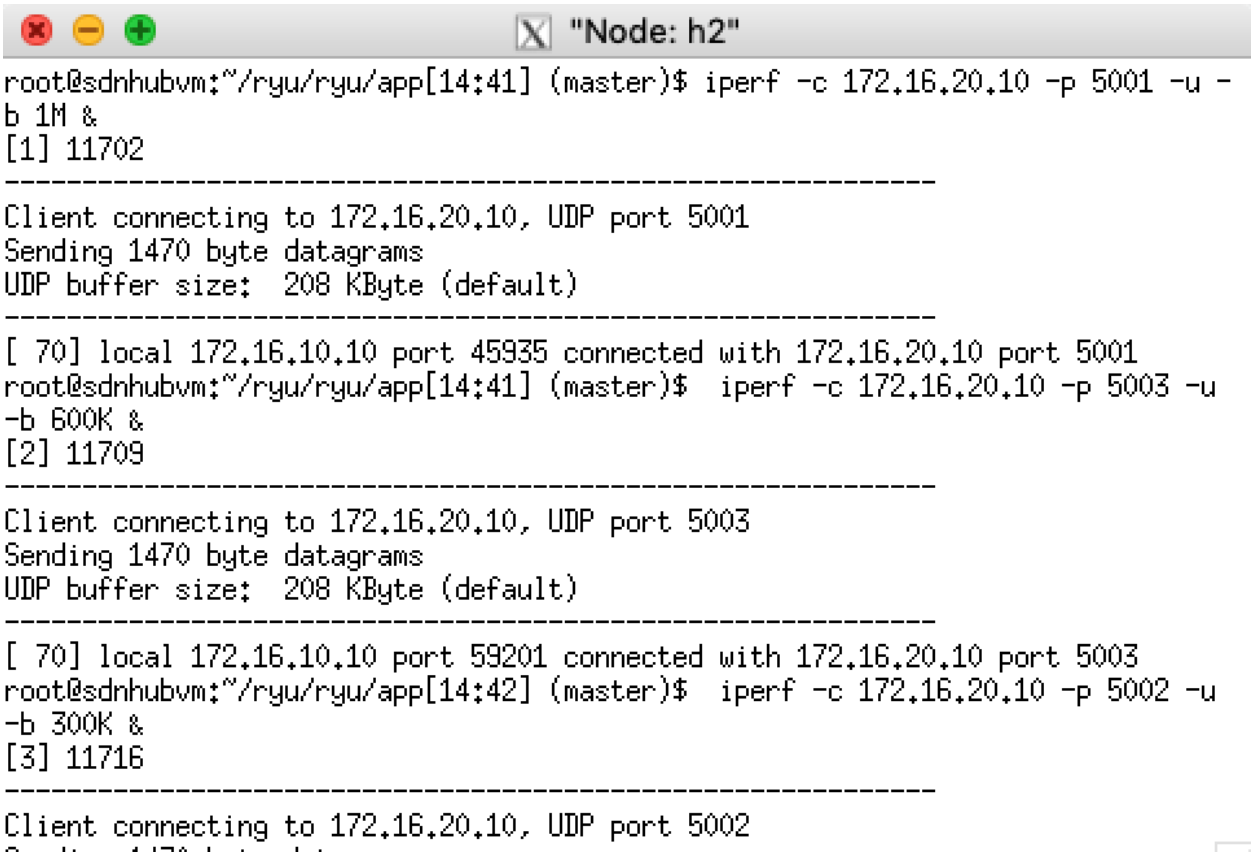


Figure A. 18 Configuring Host h2 as Client

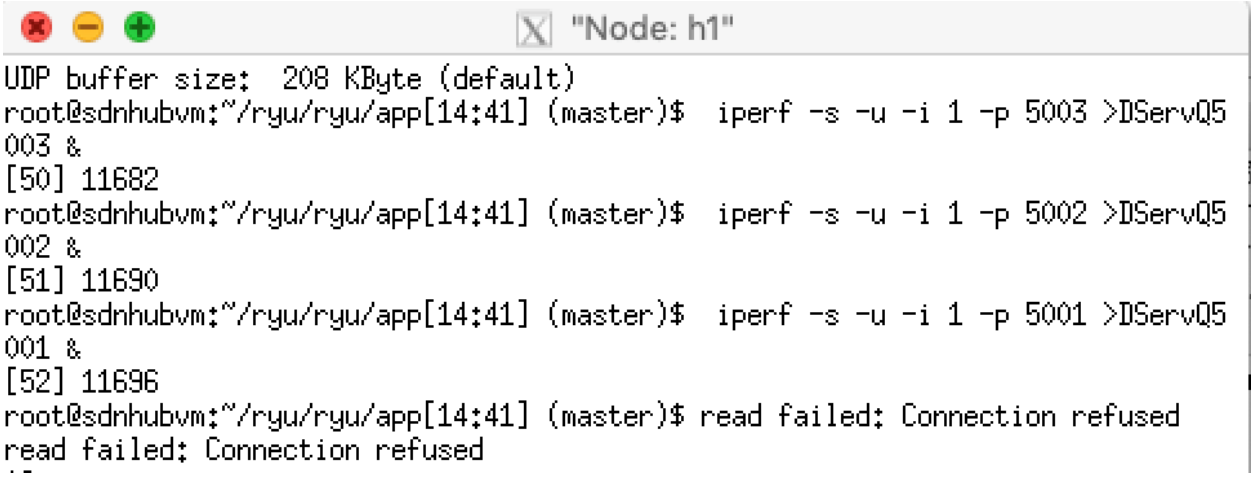


Figure A. 19 Configuring Host h1 as a Server

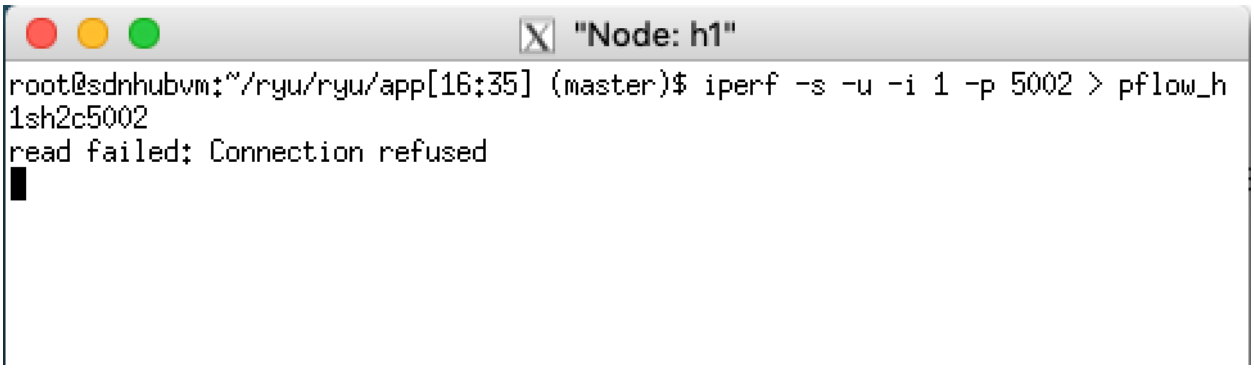


Figure A. 20 Configuring Host h1 as a Server: Per Flow QoS

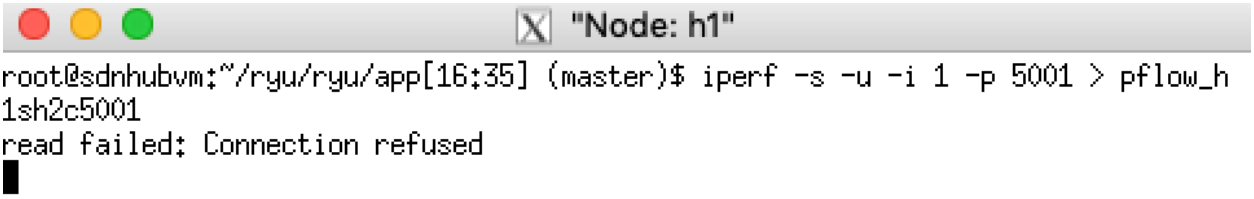


Figure A. 21 Configuring Host h1 as a Server : Per Flow QoS

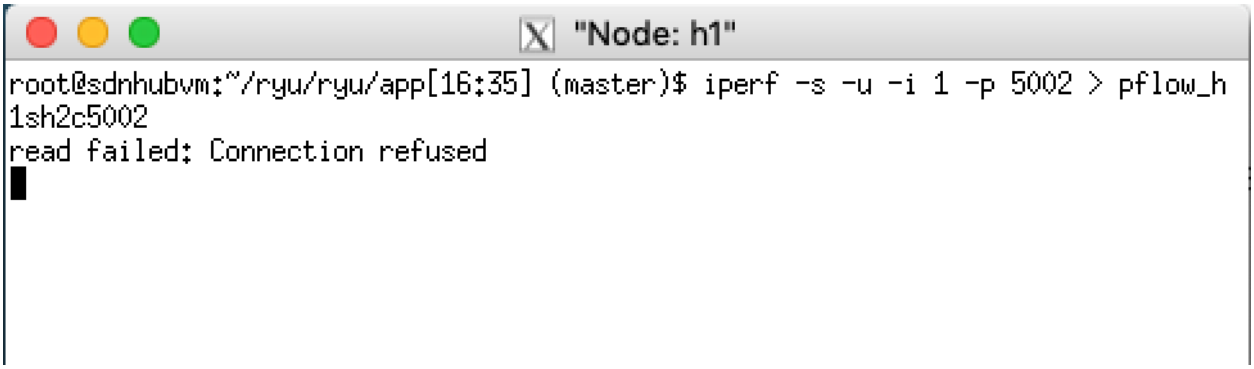


Figure A. 22 Configuring Host h1 as a Server: Per Flow QoS

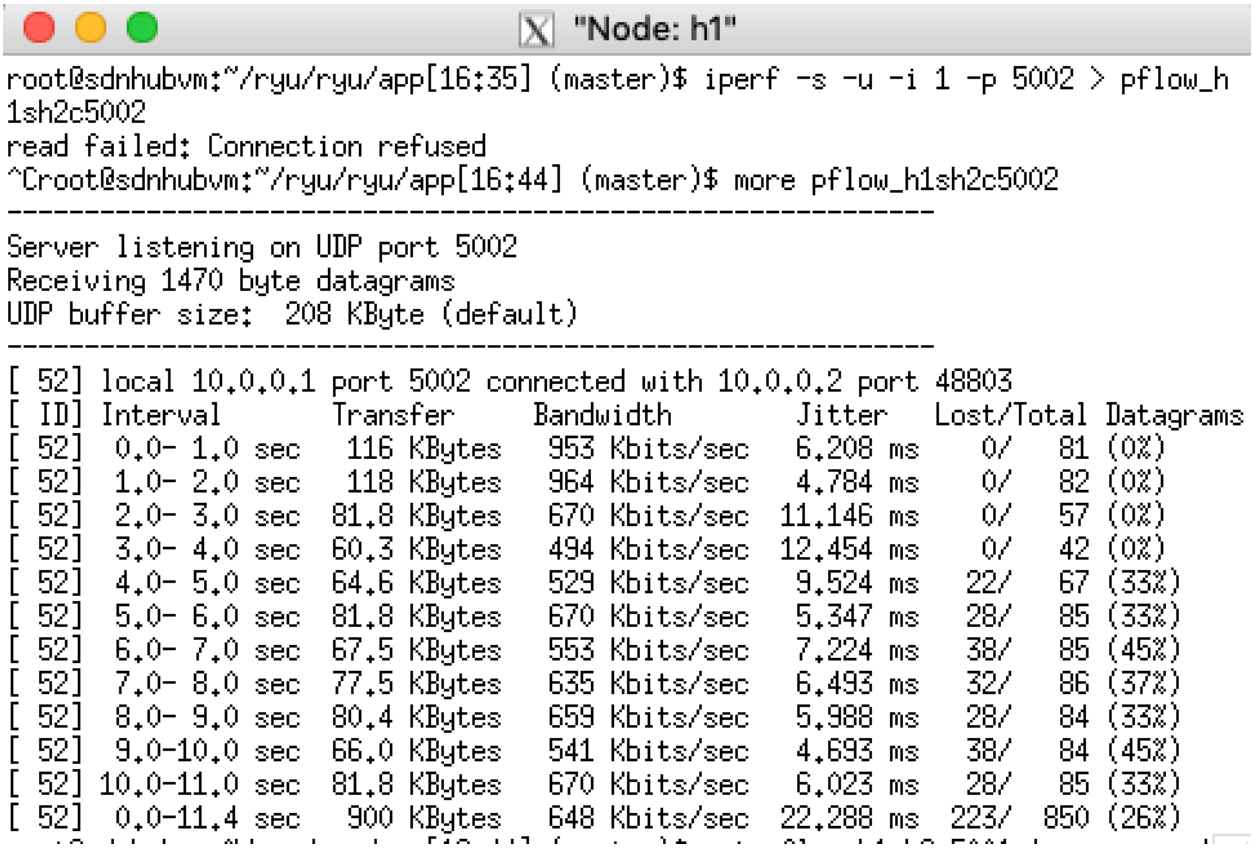


Figure A. 23 Server Listening for incoming Traffic on Port 5002: Per Flow QoS

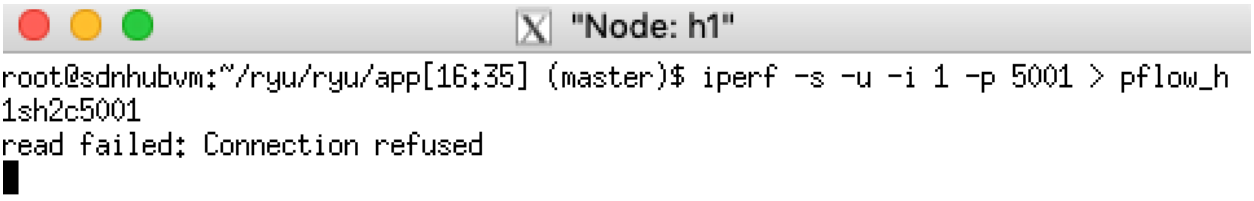


Figure A. 24 Server Listening for incoming Traffic on Port 5001: Per Flow QoS

9.8 Switch Flow Rules Verification

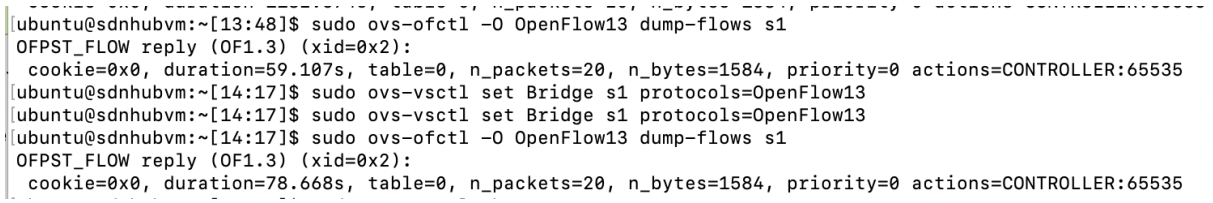


Figure A. 25 Switch S1 Flow Rules Verification

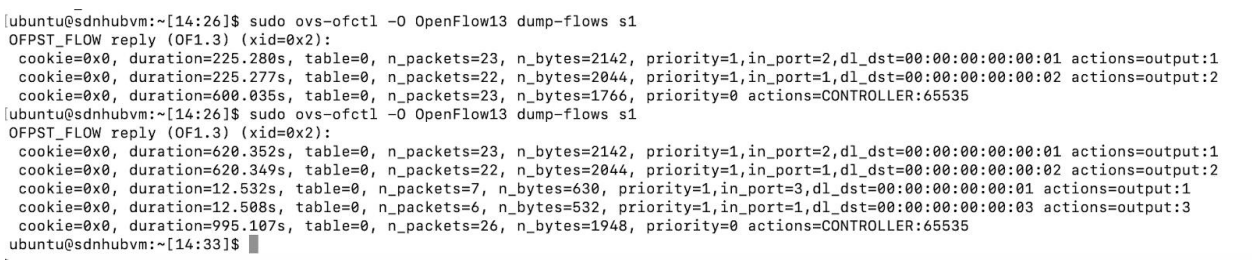


Figure A. 26 Switch S1 Flow Rules Verification

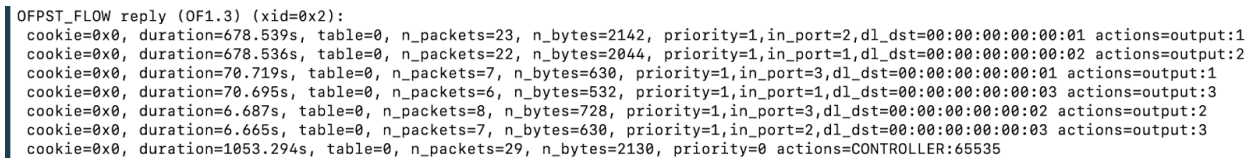


Figure A. 27 Switch Flow Rules Verification

9.9 Controller Flow Replies

```
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1        4      348        0        3      246        0
0000000000000001        2        4      348        0        4      348        0
0000000000000001 fffffffe        0        0        0        0        0        0
EVENT ofp_event->SimpleMonitor13 EventOFPPacketIn
EVENT ofp_event->SimpleMonitor13 EventOFPPacketIn
packet in 1 00:00:00:00:00:02 33:33:00:00:00:16 2
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
EVENT ofp_event->SimpleMonitor13 EventOFPPacketIn
EVENT ofp_event->SimpleMonitor13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 33:33:00:00:00:16 1
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
EVENT ofp_event->SimpleMonitor13 EventOFPPacketIn
packet in 1 00:00:00:00:00:02 33:33:00:00:00:16 2
EVENT ofp_event->SimpleMonitor13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 33:33:00:00:00:16 1
EVENT ofp_event->SimpleMonitor13 EventOFPPacketIn
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
EVENT ofp_event->SimpleMonitor13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
EVENT ofp_event->SimpleMonitor13 EventOFPPacketIn
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
EVENT ofp_event->SimpleMonitor13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
send stats request: 0000000000000001
EVENT ofp_event->SimpleMonitor13 EventOFPFlowStatsReply
EVENT ofp_event->SimpleMonitor13 EventOFPPortStatsReply
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1        9      738        0       13     1026        0
0000000000000001        2        9      738        0       15     1206        0
0000000000000001 fffffffe        0        0        0        0        0        0
send stats request: 0000000000000001
EVENT ofp_event->SimpleMonitor13 EventOFPFlowStatsReply
EVENT ofp_event->SimpleMonitor13 EventOFPPortStatsReply
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1        9      738        0       13     1026        0
0000000000000001        2        9      738        0       15     1206        0
0000000000000001 fffffffe        0        0        0        0        0        0
send stats request: 0000000000000001
EVENT ofp_event->SimpleMonitor13 EventOFPFlowStatsReply
EVENT ofp_event->SimpleMonitor13 EventOFPPortStatsReply
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
```

Figure A. 28 Ryu Controller Replies: Packets

```
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1       12      920        0       16     1208        0
0000000000000001        2       12      920        0       18     1388        0
0000000000000001 fffffffe        0        0        0        0        0        0
send stats request: 0000000000000001
EVENT ofp_event->SimpleMonitor13 EventOFPFlowStatsReply
EVENT ofp_event->SimpleMonitor13 EventOFPPortStatsReply
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
0000000000000001        1 00:00:00:00:00:02        2        2      140
0000000000000001        2 00:00:00:00:00:01        1        3      182
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1       12      920        0       16     1208        0
0000000000000001        2       12      920        0       18     1388        0
0000000000000001 fffffffe        0        0        0        0        0        0
send stats request: 0000000000000001
EVENT ofp_event->SimpleMonitor13 EventOFPFlowStatsReply
EVENT ofp_event->SimpleMonitor13 EventOFPPortStatsReply
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
0000000000000001        1 00:00:00:00:00:02        2        2      140
0000000000000001        2 00:00:00:00:00:01        1        3      182
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1       12      920        0       16     1208        0
0000000000000001        2       12      920        0       18     1388        0
0000000000000001 fffffffe        0        0        0        0        0        0
send stats request: 0000000000000001
EVENT ofp_event->SimpleMonitor13 EventOFPFlowStatsReply
EVENT ofp_event->SimpleMonitor13 EventOFPPortStatsReply
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
0000000000000001        1 00:00:00:00:00:02        2        2      140
0000000000000001        2 00:00:00:00:00:01        1        3      182
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1       12      920        0       16     1208        0
0000000000000001        2       12      920        0       18     1388        0
0000000000000001 fffffffe        0        0        0        0        0        0
send stats request: 0000000000000001
EVENT ofp_event->SimpleMonitor13 EventOFPFlowStatsReply
EVENT ofp_event->SimpleMonitor13 EventOFPPortStatsReply
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
0000000000000001        1 00:00:00:00:00:02        2        2      140
0000000000000001        2 00:00:00:00:00:01        1        3      182
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1       12      920        0       16     1208        0
0000000000000001        2       12      920        0       18     1388        0
0000000000000001 fffffffe        0        0        0        0        0        0
```

Figure A. 29 Ryu Controller Replies: Packets

9.10 QoS Configuration Settings

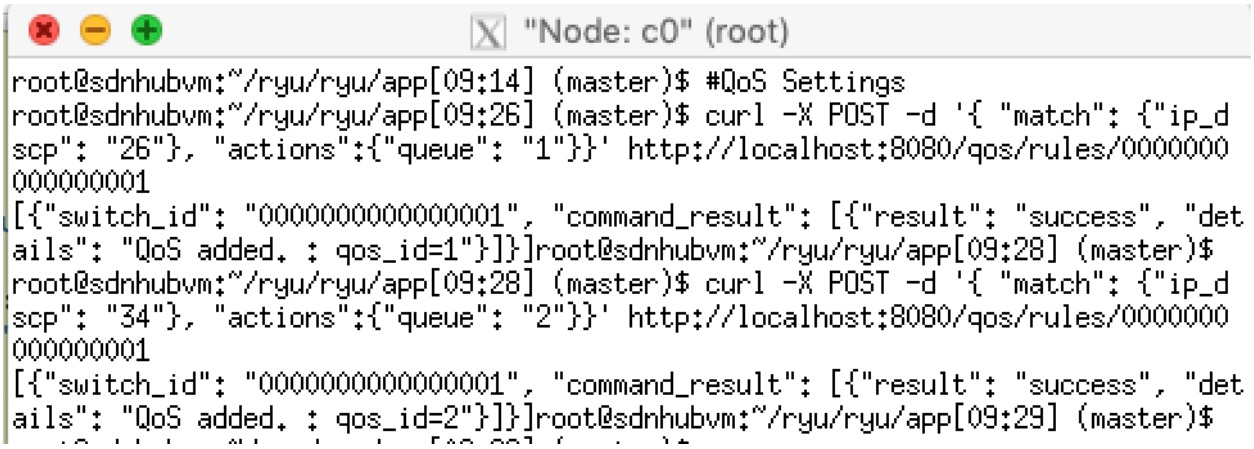


Figure A. 30 Ryu Controller QoS Settings Configuration

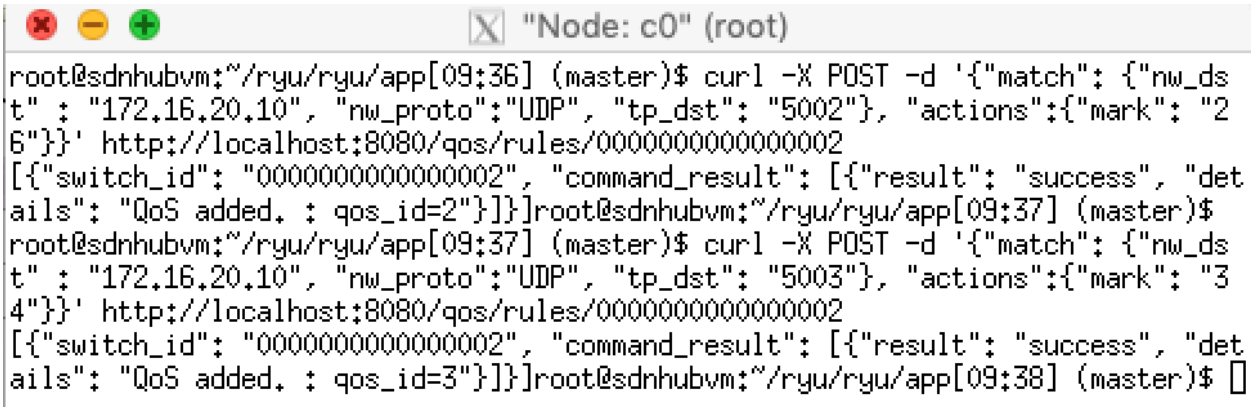


Figure A. 31 Ryu Controller QoS Settings Configuration

9.11 Cat Linux Commands to Extract Data

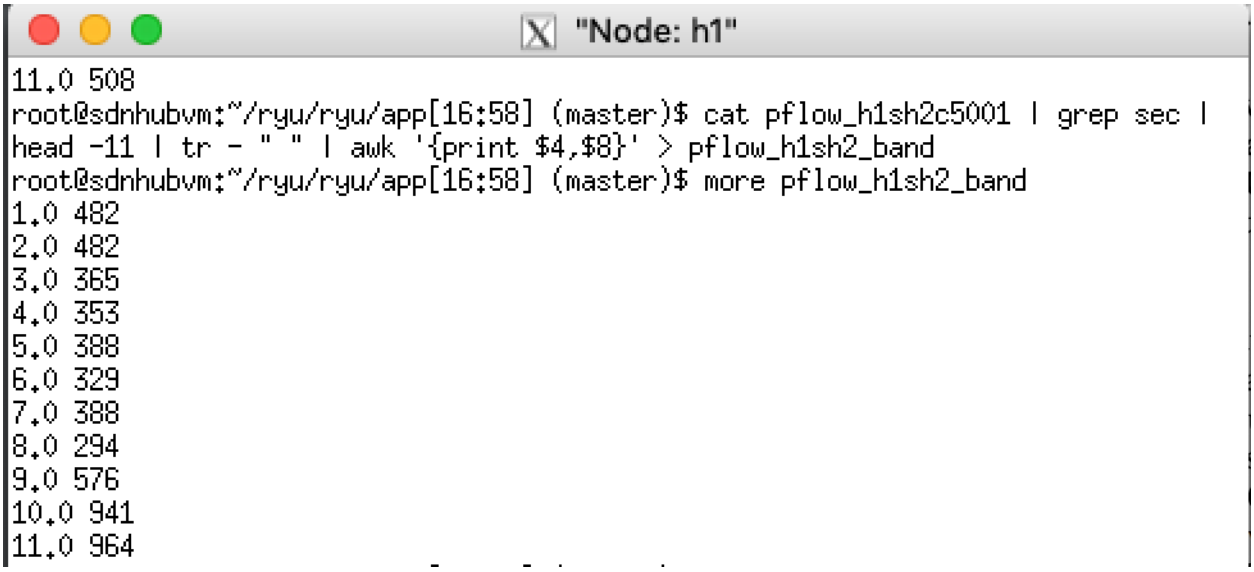


Figure A. 32 cat command to extract bandwidth - time information

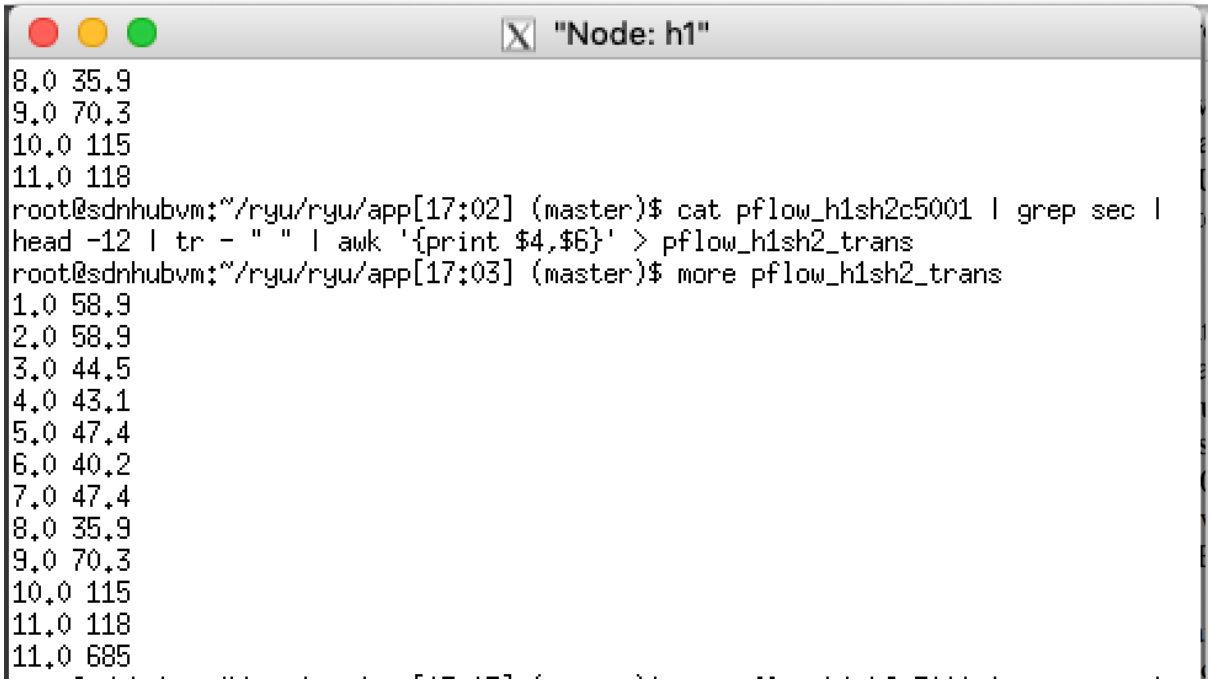


Figure A. 33 cat command to extract datagram transfer size - time information

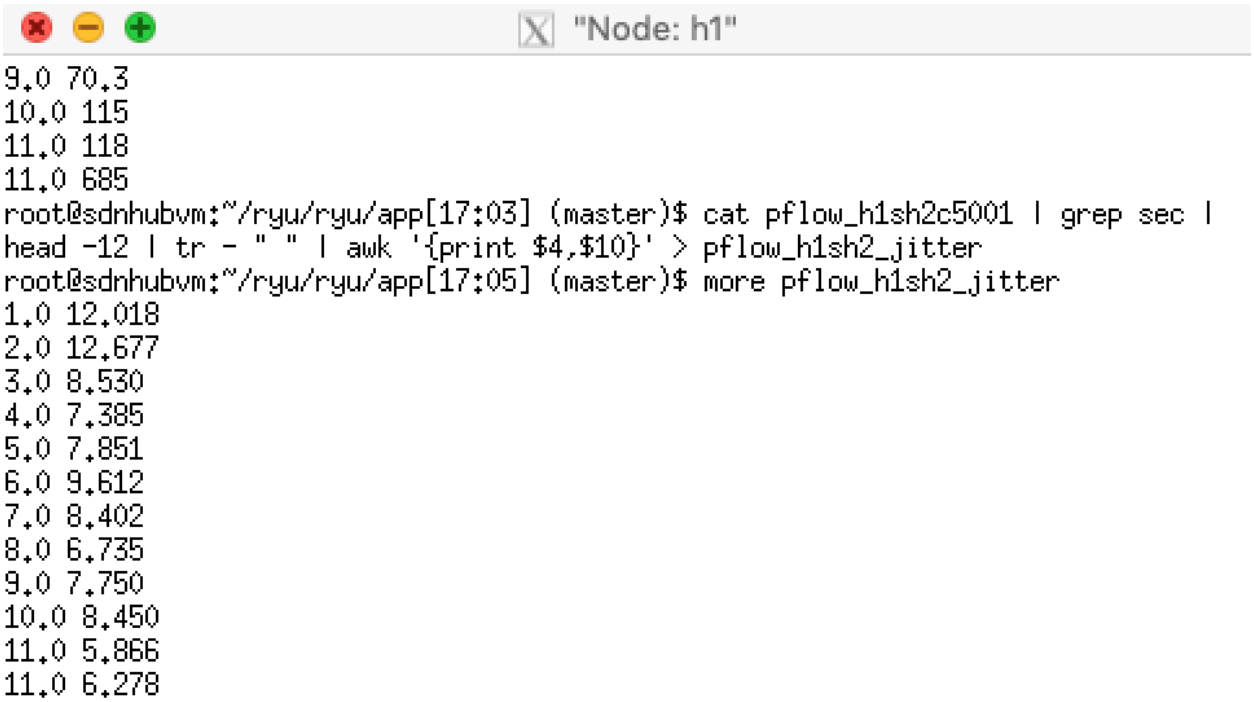


Figure A. 34 cat command to extract jitter - time information

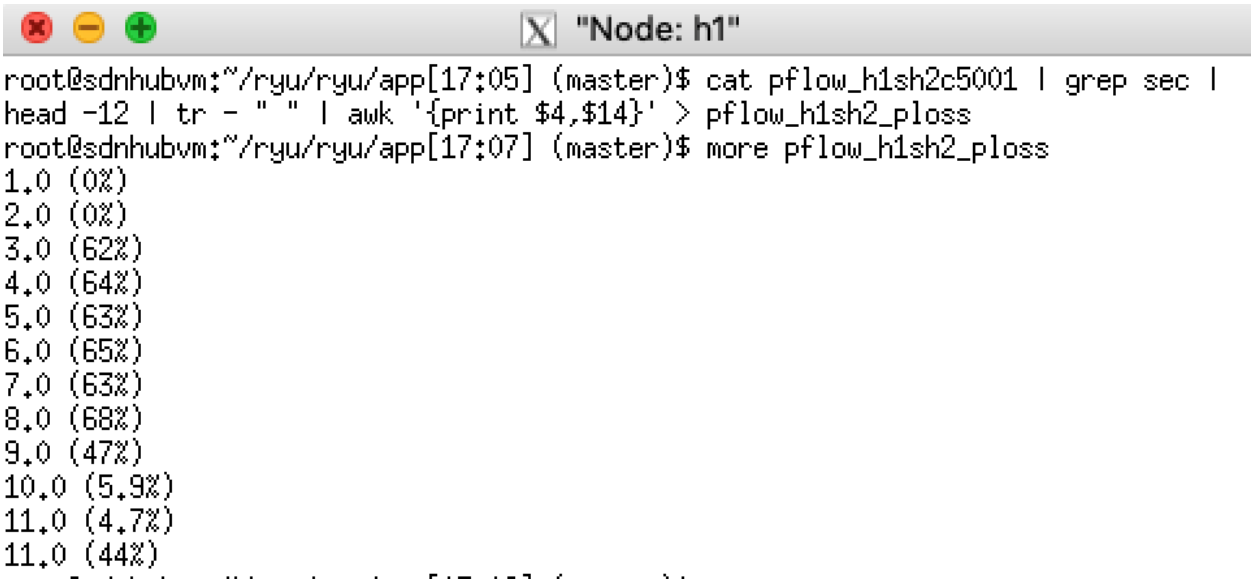


Figure A. 35 cat command to extract datagram percent loss - time information

9.12 GNUPLOT Commands

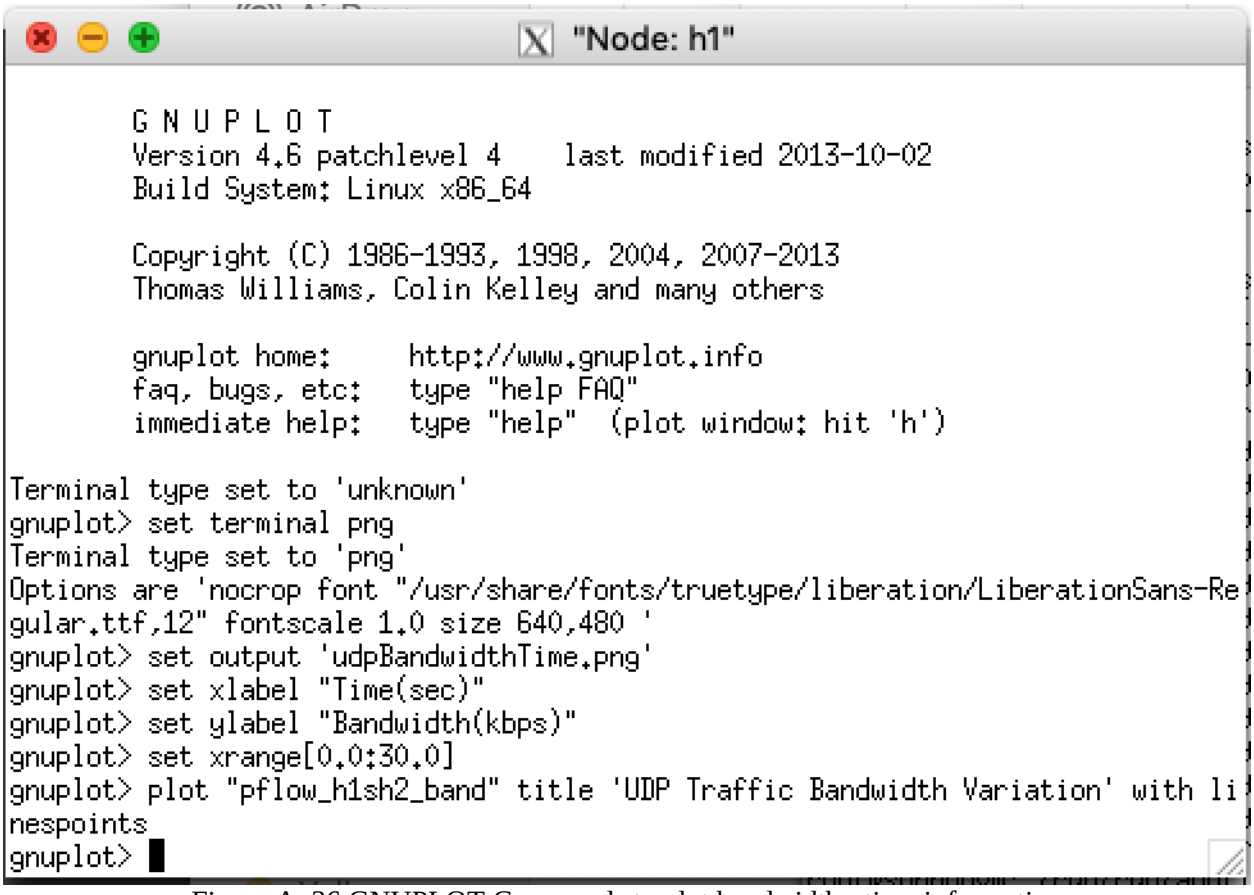


Figure A. 36 GNUPLOT Commands to plot bandwidth - time information

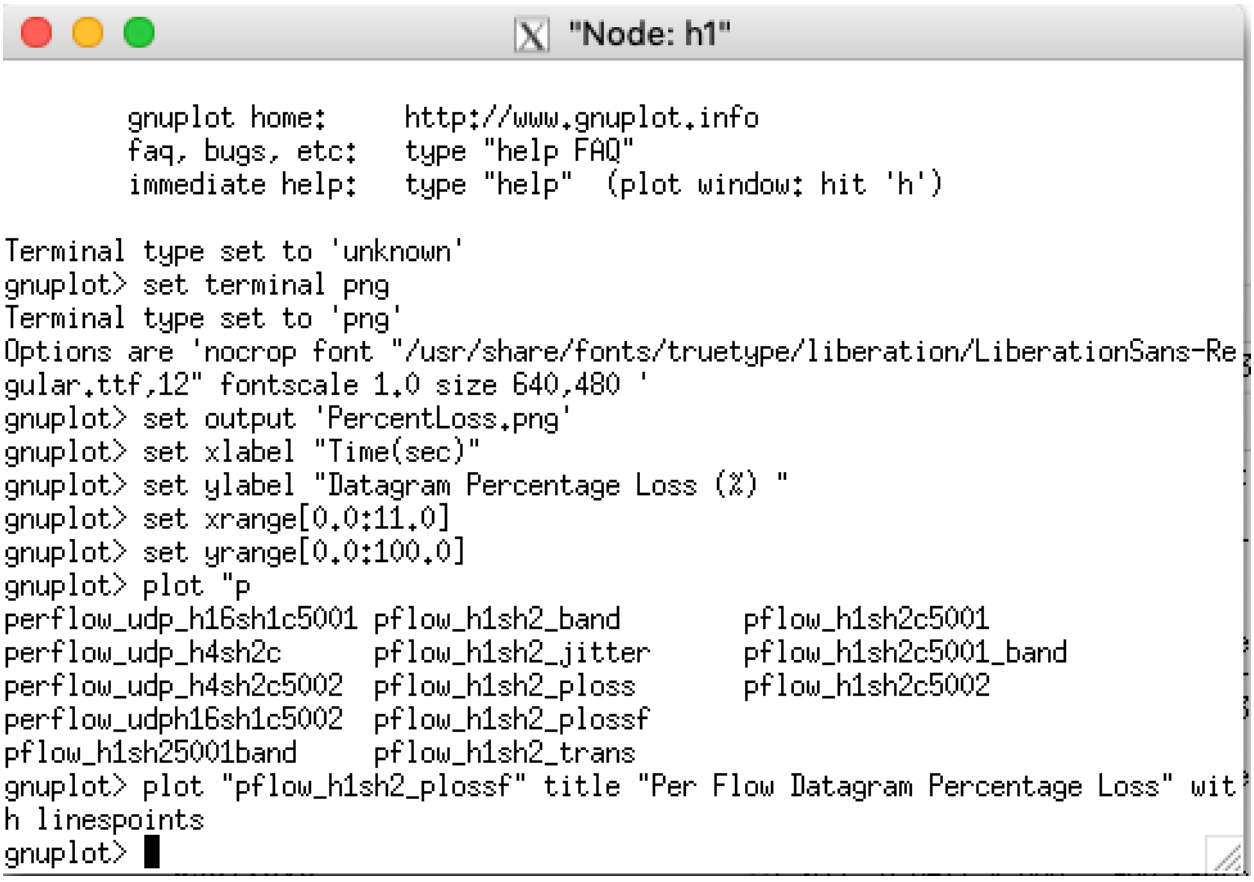


Figure A. 37 GNUPLOT Commands to plot datagram percent loss - time information

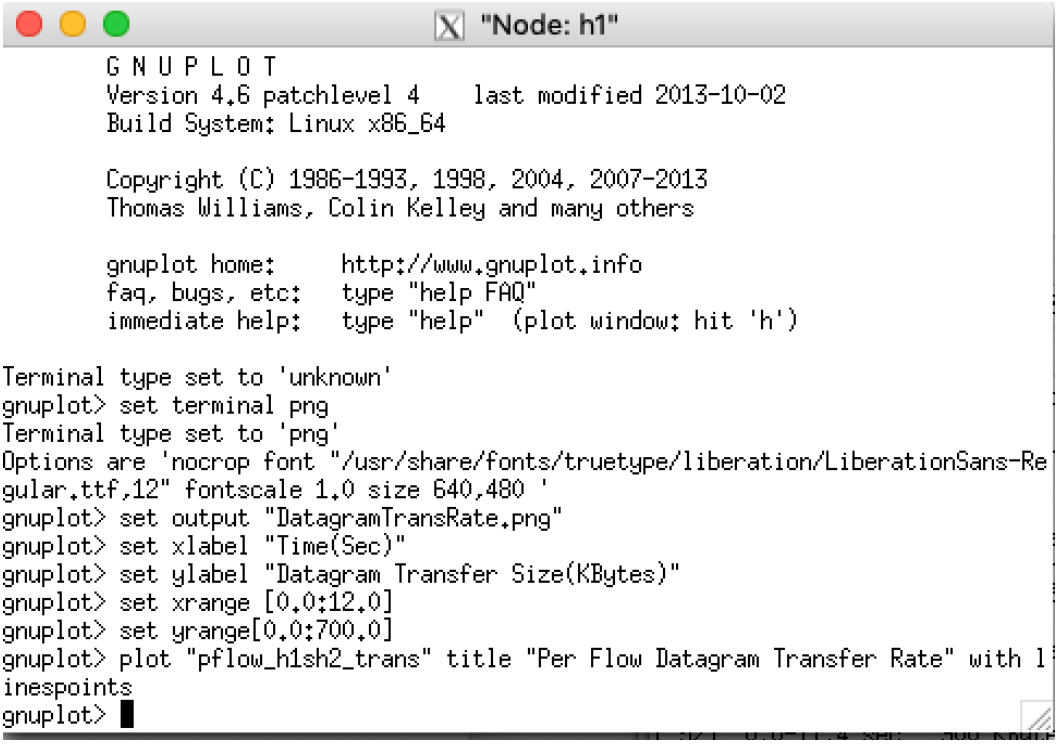


Figure A. 38 GNUPLOT Commands to plot datagram transfer - time information

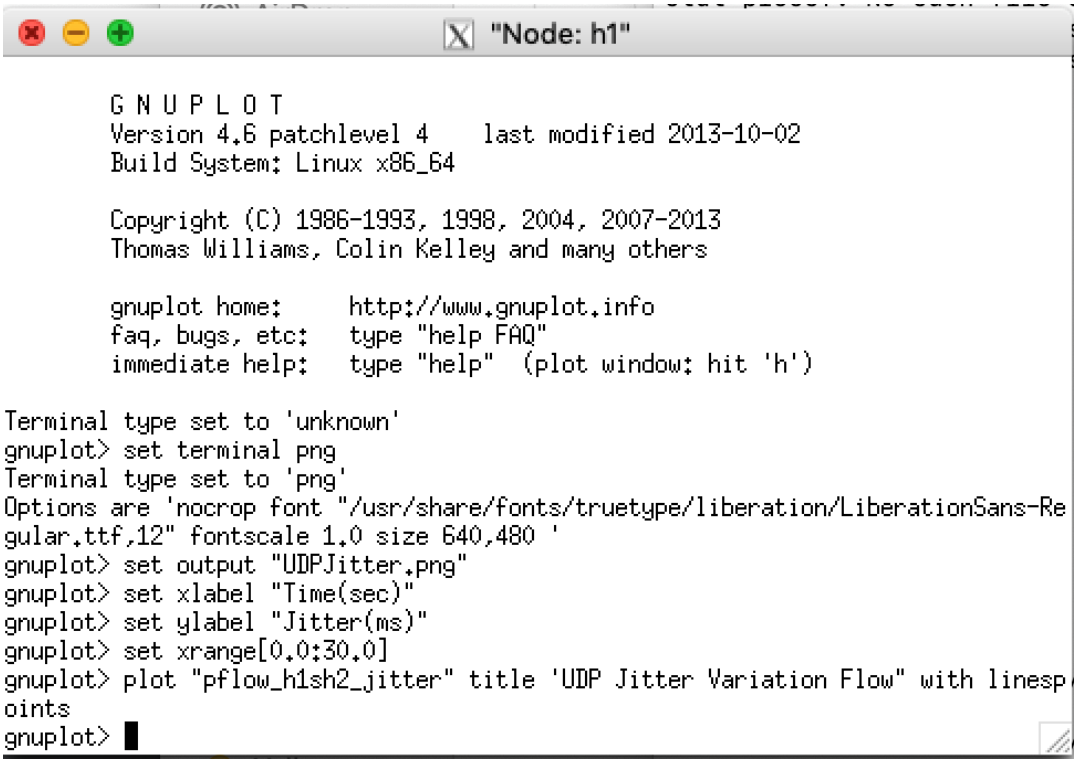


Figure A. 39 GNUPLOT Commands to plot jitter - time information

9.2. Appendix B. 1st Model: Tree Topology Network Python Code

```
#MEng Thesis
#Date : 23 September 2019
#Author : Lindokuhle Biyase
#Institution : UCT


from mininet.net import Mininet
from mininet.cli import CLI
from mininet.topo import Topo
from mininet.node import UserSwitch
from mininet.node import RemoteController


class SliceableSwitch(UserSwitch):
    def __init__(self, name, **kwargs):
        UserSwitch.__init__(self, name, '', **kwargs)


class MyTopo(Topo):
```

```
def __init__( self ):
    "Create custom topo."
    # Initialize topology
    Topo.__init__( self )
    # Add hosts and switches
    host01 = self.addHost('h1')
    host02 = self.addHost('h2')
    host03 = self.addHost('h3')


    host04 =self.addHost('h4')
    host05 = self.addHost('h5')
    host06 =self.addHost('h6')
    host07 =self.addHost('h7')
    host08=self.addHost('h8')


    host09 = self.addHost('h9')
    host10 = self.addHost('h10')
    host11 = self.addHost('h11')
    host12 = self.addHost('h12')


    host13 = self.addHost('h13')
    host14 = self.addHost('h14')
    host15 = self.addHost('h15')
    host16 = self.addHost('h16')


    switch01 = self.addSwitch('s1')
    switch02 = self.addSwitch('s2')
    switch03 = self.addSwitch('s3')
    switch04 = self.addSwitch('s4')
    switch05 = self.addSwitch('s5')
    switch06 = self.addSwitch('s6')
    switch07 = self.addSwitch('s7')
    # Add links
    #Switches
   # self.addLink(switch01, switch02)
    self.addLink(switch01,switch04)
    self.addLink(switch01, switch05)
    self.addLink(switch01, switch06)
    self.addLink(switch01, switch07)
   # self.addLink(switch01, switch02)
    self.addLink(switch01, switch02)
    self.addLink(switch01, switch03)


    #Switch s2
    self.addLink(switch02,switch04)
    self.addLink(switch02, switch05)
    self.addLink(switch02, switch06)
```

```
        self.addLink(switch02, switch07)
        # self.addLink(switch02, switch01)
        self.addLink(switch02, switch03)


        #Switch s3
        self.addLink(switch03, switch04)
        self.addLink(switch03,switch05)
        self.addLink(switch03, switch06)
        self.addLink(switch03, switch07)
        # self.addLink(switch03, switch01)
        # self.addLink(switch03, switch02)
        #Switch04 -->Host1-4
        self.addLink(host01, switch04)
        self.addLink(host02, switch04)
        self.addLink(host03, switch04)
        self.addLink(host04, switch04)


        #Switch05 -Host5-8
        self.addLink(host05, switch05)
        self.addLink(host06, switch05)
        self.addLink(host07, switch05)
        self.addLink(host08, switch05)


        #Switch 06 - Host 9-12
        self.addLink(host09,switch06)
        self.addLink(host10,switch06)
        self.addLink(host11,switch06)
        self.addLink(host12, switch06)

        #Switch 07 -- Host 13 - 16
        self.addLink(host13, switch07)
        self.addLink(host14, switch07)
        self.addLink(host15, switch07)
        self.addLink(host16, switch07)


        # self.addLink(host01, switch01)
        #self.addLink(host02, switch02)
        # self.addLink(host03, switch03)
        # self.addLink(switch01, switch02)
        #self.addLink(switch01, switch03)


def run(net):
    s1 = net.getNodeByName('s1')
    s1.cmdPrint('dpctl unix:/tmp/s1 queue-mod 1 1 80')
    s1.cmdPrint('dpctl unix:/tmp/s1 queue-mod 1 2 120')
    s1.cmdPrint('dpctl unix:/tmp/s1 queue-mod 1 3 800')
```

```
def genericTest(topo):
    net = Mininet(topo=topo, switch=SliceableSwitch,
        controller=RemoteController)
    net.start()
    run(net)
    CLI(net)
    net.stop()


def main():
    topo = MyTopo()
    genericTest(topo)


if __name__ == '__main__' :
    main()
```

9.3. Appendix B. 1st Model: Linear Topology Network Python Code

```
#MEng Thesis
#Date : 27 September 2019
#Author : Lindokuhle Biyase
#Linear Topology
#Institution : UCT


from mininet.net import Mininet
from mininet.cli import CLI
from mininet.topo import Topo
from mininet.node import UserSwitch
from mininet.node import RemoteController


class SliceableSwitch(UserSwitch):
    def __init__(self, name, **kwargs):
        UserSwitch.__init__(self, name, '', **kwargs)


class MyTopo(Topo):
    def __init__( self ):
        "Create custom topo."
        # Initialize topology
        Topo.__init__( self )
        # Add hosts and switches
        host01 = self.addHost('h1')
        host02 = self.addHost('h2')
        host03 = self.addHost('h3')
```

```
host04 =self.addHost('h4')
host05 = self.addHost('h5')
host06 =self.addHost('h6')
host07 =self.addHost('h7')
host08=self.addHost('h8')


host09 = self.addHost('h9')
host10 = self.addHost('h10')
host11 = self.addHost('h11')
host12 = self.addHost('h12')


host13 = self.addHost('h13')
host14 = self.addHost('h14')
host15 = self.addHost('h15')
host16 = self.addHost('h16')


switch01 = self.addSwitch('s1')
switch02 = self.addSwitch('s2')
switch03 = self.addSwitch('s3')
switch04 = self.addSwitch('s4')
switch05 = self.addSwitch('s5')
switch06 = self.addSwitch('s6')
switch07 = self.addSwitch('s7')
switch08 = self.addSwitch('s8')
switch09 = self.addSwitch('s9')
switch10 = self.addSwitch('s10')
switch11 = self.addSwitch('s11')
switch12 = self.addSwitch('s12')
switch13 = self.addSwitch('s13')
switch14 = self.addSwitch('s14')
switch15 = self.addSwitch('s15')
switch16 = self.addSwitch('s16')


# Add links
self.addLink(host01, switch01)
self.addLink(host02, switch02)
self.addLink(host03, switch03)
self.addLink(host04, switch04)
self.addLink(host05, switch05)
self.addLink(host06, switch06)
self.addLink(host07, switch07)
self.addLink(host08, switch08)
self.addLink(host09, switch09)
self.addLink(host10, switch10)
self.addLink(host12, switch12)
self.addLink(host11, switch11)
```

```
        self.addLink(host13, switch13)
        self.addLink(host14, switch14)
        self.addLink(host15, switch15)
        self.addLink(host16, switch16)


        self.addLink(switch01,switch02)
        self.addLink(switch02,switch03)
        self.addLink(switch03,switch04)
        self.addLink(switch04,switch05)
        self.addLink(switch05,switch06)
        self.addLink(switch06,switch07)
        self.addLink(switch07,switch08)
        self.addLink(switch08,switch09)
        self.addLink(switch09,switch10)
        self.addLink(switch10,switch11)
        self.addLink(switch11,switch12)
        self.addLink(switch12,switch13)
        self.addLink(switch13,switch14)
        self.addLink(switch14,switch15)
        self.addLink(switch15,switch16)


def run(net):
    s1 = net.getNodeByName('s1')
    s1.cmdPrint('dpctl unix:/tmp/s1 queue-mod 1 1 80')
    s1.cmdPrint('dpctl unix:/tmp/s1 queue-mod 1 2 120')
    s1.cmdPrint('dpctl unix:/tmp/s1 queue-mod 1 3 800')


def genericTest(topo):
    net = Mininet(topo=topo, switch=SliceableSwitch,
        controller=RemoteController)
    net.start()
    run(net)
    CLI(net)
    net.stop()


def main():
    topo = MyTopo()
    genericTest(topo)


if __name__ == '__main__' :
    main()
```