



**DEBRE BERHAN UNIVERSITY**

**COLLEGE OF COMPUTING**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**PERFORMANCE ENHANCEMENT OF RYU CONTROLLER BASED-  
SOFTWARE DEFINED NETWORKS USING MPLS**

**A Thesis Submitted to College of Computing in Partial Fulfillment of the  
Requirements for Master of Sciences in Computer Networks and Security**

**HABTAMU GEBEYEHU ESHETU**

**January, 2024 G.C**

**Debre Berhan, Ethiopia**

DEBRE BERHAN UNIVERSITY  
COLLEGE OF COMPUTING  
DEPARTMENT OF INFORMATION TECHNOLOGY

**Performance Enhancement of RYU Controller Based- Software Defined  
Networks Using MPLS**

A Thesis Submitted to College of Computing in Partial Fulfillment of the  
Requirements for Master of Sciences in Computer Networks and Security

HABTAMU GEBEYEHU ESHETU

**Advisor: Samuel Asferaw (Ph.D)**

January, 2024 G.C

Debre Berhan, Ethiopia

**DEBRE BERHAN UNIVERSITY**

**COLLEGE OF COMPUTING**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**BY: HABTAMU GEBEYEHU ESHETU**

**ADVISOR: Dr. SAMUEL ASFERAW**


The final reading approval of the thesis prepared by **Habtamu Gebeyehu Eshetu**, titled: Performance Enhancement of RYU Controller Based- Software Defined Networks Using MPLS and Submitted to the Department of Information Technology in Partial Fulfillment of the Requirements for Master of Sciences in Computer Networks and Security with the regulations of the university and meets the accepted standards with respect to originality and quality.

**Approved By:**

Advisor: Dr. Samuel Asferaw

Signature-----Date.....

External Examiner: Dr. Asrat Mulatu

  
Signature-----Date.....22/01/2024

Internal Examiner: Assistant Prof. Alebachew Chiche Signature----- Date.....

Chairperson: -----

Signature-----Date.....

## DECLARATION

I, the undersigned, declare that this thesis entitled “Performance Enhancement of RYU Controller Based- Software Defined Networks Using MPLS” is my original work and has not been presented for a degree in this or any other universities, and all sources of references used for the thesis work have been appropriately acknowledged.

Name: Habtamu Gebeyehu Eshetu

Signature: \_\_\_\_\_ Date: \_\_\_\_\_

The thesis has been submitted for examination with my approval as university advisor.

Advisor Name: Samuel Asferaw (Ph.D)

Signature: \_\_\_\_\_ Date: \_\_\_\_\_

## ACKNOWLEDGEMENTs

First and foremost, I would like to thank my God for giving me the strength to carry out and complete this work.

I would like to express my special thanks and appreciation to my advisor **Dr. Samuel Asferaw**, for his patient guidance, support, encouragement, helpful advice, and giving me the opportunity for doing what turned out to be exciting Master's thesis. He's technical and editorial advice was essential to the completion of this dissertation and has taught me infinite lessons.

I would also like to thank my parents, sisters and my friends especially Mr. Seifu Alemayehu and his wife, Mr. Ashneafi Tadesse and his wife, Mr. Robel Cheber and his wife, Mr. Nebiyat, Mr. Behailu Korma, Mr. Fasil Tsegaye, Dr. Esubalew Getenet, W/rt Jale Obelika, Mr. Shetaw Tafesse and his wife, Mr. Ibrahim Neji and his wife, Mr. Getasew Alemayehu, Mr. Mengistu Muluken and his wife and all of Ethernet staff IN Ministry of Education specially Mr. Yonatan Mekuriyaw, and Mr. Eyuel for his kind and continuous support. Many thanks also go to my dear friends for their continuous encouragement and support throughout my thesis. I wish to extend my gratitude for my wife and my child who are my blessing for their unconditional love for their support, patience and love. Without their encouragement, motivation and understanding, it would have been impossible for me to complete this work. Finally, thanks to all people who supported me to complete this work.

Habtamu Gebeyehu

## ABSTRACT

Software Defined Networking (SDN) is a rapidly growing technology that offers open and flexible networking solutions through the management of virtual devices using a mininet network emulator. It simplifies network administration by centralizing control and separating the control plane from the data plane. One of the major research areas in SDN is performance optimization, which involves considering three parameters and their dependencies. The research gap by implementing using GNS3 and Mininet a performance enhancement of the Ryu controller mechanism using Multiprotocol Label Switching (MPLS). The study utilized the Mininet network emulator to create a network environment consisting of a virtual router and hosts controlled by the Ryu controller.

The network system is followed by performance evaluation using the Cbench tool, which measures throughput, latency and jitter metrics. To effectively distribute network loads, MPLS used instead of a single controller. This thesis approach influenced MPLS networks to handle low-level packet flows, allowed for efficient traffic management. The evaluation results indicated that the integration of MPLS and load distribution using the Ryu controller led to significant improvements in network performance. Based on the findings obtained from the research conducted, it appears that integrating MPLS (Multi-Protocol Label Switching) with the Ryu controller in an SDN environment resulted in performance enhancements. Specifically, the average throughput was improved by 4% compared to without MPLS on the data plane; and the latency was reduced by 5%. Based on the findings of the thesis, it was determined that combining MPLS and SDN is an effective way to increase the overall performance of software-defined networks compared to the normal SDN or without MPLS implementation.

**Keywords:** *Software Defined Networking (SDN), Open Flow, Ryu Controller, Multiprotocol Label Switching (MPLS), GNS3, Cbench, Throughput, Latency, Jitter.*

## Table of Contents

DECLARATION .....	III
ACKNOWLEDGEMENTs.....	IV
ABSTRACT.....	V
List of Figures .....	VIII
List of Tables .....	IX
List of Abbreviations and Acronyms.....	X
Chapter One: Introduction .....	1
1.1 Background of the Study.....	1
1.2 Statement of the Problem .....	2
1.3 Objectives of the Study .....	4
1.3.1 General Objective .....	4
1.3.2 Specific Objectives .....	4
1.4 Scope of the Study.....	4
1.5 Significance of the Study .....	4
1.6 Organization of the Thesis .....	5
Chapter Two: Literature Review .....	6
2.1 Introduction .....	6
2.2 Overview of SDN Architecture.....	6
2.2.1 SDN Northbound Interface.....	8
2.2.2 SDN Southbound Interface.....	8
2.3 Ryu Architecture .....	9
2.3.1Traditional Network Architecture .....	10
2.3.2 Limitations of Traditional Networks .....	12
2.4 Open Flow Protocol .....	13
2.5 Packet Processing Mechanism .....	15
2.6 Taxonomy of Load Balancing Approaches.....	15
2.7 Load balancing in SDN.....	17
2.8 Types of load-balancing algorithms: Static versus Dynamic.....	18
2.9 MPLS Architecture .....	19

2.9.1 MPLS Process using LSR.....	20
2.9.2 Benefits of MPLS networks .....	22
2.10 Related works .....	24
2.11 Gap Analysis from Related Works.....	28
Chapter Three: The Proposed Enhanced RYU controller using MPLS .....	29
3.1. Introduction .....	29
3.2 RYU Controller .....	29
3.3 Research Methodology.....	30
3.4 Cbench Algorithm .....	31
3.5 Proposed Method.....	31
3.6 Building blocks of MPLS.....	35
Chapter Four: Implementation, Result and Analysis .....	37
4.1 Setup of the simulation Environment.....	37
4.2 Topology .....	37
4.3 Simulation Tools and Techniques .....	38
4.3.1 Installation Ryu controller .....	38
4.3.2 Installation and version.....	39
4.4 Message establishment between a switch and a controller .....	42
4.5 Messages exchanged between two hosts.....	42
4.6 Performance Measuring Metrics .....	43
4.6.1 Throughput .....	44
4.6.2 Latency .....	49
4.6.3 Jitter .....	52
Chapter Five: Conclusions and Future Works .....	57
5.1 Conclusions .....	57
5.2 Future Works.....	58
References.....	59
Appendix.....	63



## List of Figures

Figure 2.1: The three layers in SDN architecture .....	7
Figure 2.2 : RYU SDN controller architecture .....	10
Figure 2.3 : Legacy network architecture .....	11
Figure 2.4 : OpenFlow table entries.....	13
Figure 2.5: Open Flow architecture .....	14
Figure 2.6: Packet flow in Open flow switch .....	15
Figure 2.7: Taxonomy of Load Balancing Approaches.....	17
Figure 2.8: General MPLS Architecture [41] .....	20
Figure 2.9: Label Insertion [41].....	21
Figure 2.10: Label Swapping [41] .....	22
Figure 3.1: General Methods using MPLS .....	32
Figure 3.2 Detail Flowchart Ryu Controller with MPLS network .....	34
Figure 4.1: GNS3 with Mininet Topology.....	38
Figure 4.2: Ryu-manager version .....	39
Figure 4.3: Mininet Versions .....	39
Figure 4.4: Python version.....	39
Figure 4.5: Open vSwitch and Database Schema Version .....	39
Figure 4.6: linear topology Installation.....	40
Figure 4.7: linear topology Installation 32 switches.....	40
Figure 4.8: Open vSwitch connections .....	41
Figure 4.9: Mininet Installation .....	42
Figure 4.10: Wireshark testing from h1 to h2.....	43
Figure 4.11: Cbench Throughput Result without MPLS one switch.....	46
Figure 4.12: Number of switch Tests with and without MPLS Ryu controller .....	47
Figure 4.13: Cbench Throughput Result with MPLS one switch.....	47
Figure 4.14: Throughput of proposed method with MPLS and normal SDN .....	48
Figure 4.15: Cbench Latency Result without MPLS one switch.....	50
Figure 4.16: Cbench Latency switch tests with and without MPLS result.....	50
Figure 4.17: Latency of Ryu controller with MPLS .....	51
Figure 4.18: Latency Result proposed method and Normal SDN .....	51
Figure 4.19 : Jitter result h1 to h2 .....	53
Figure 4.20: Jitter result h1 to h4.....	53
Figure 4.21: Jitter Result Bandwidth vs Jitter.....	54

## List of Tables

Table 2.1: Gap Analysis of Related works .....	28
Table 3.1: Cbench Algorithm .....	31
Table 4.1: Cbench Running Options.....	44
Table 4.2: Cbench Throughput result with and without MPLS network (Normal SDN).....	46
Table 4.3: Latency total result with and without MPLS Network.....	49
Table 4.4: Jitter Running Options.....	53
Table 4.5: Result Jitter single topology .....	54
Table 4.6: Shows the overall performance metrics used in this research study as compared to previous works .....	55

## **List of Abbreviations and Acronyms**

SDN	Software Defining Network
OF	Open Flow
LB	Load Balancer
CPU	Central Processing Unit
PC	Personal Computer
IP	Internet Protocol
ODL	Open Day Light
API	Application Programming Interface
IoT	Internet of Things
QoS	Quality of Service
SDN	Software Defined Networking
NFV	Network Function Virtualization
ACLs	Access Lists
ISP	Internet service providers
MPLS	Multiprotocol Label Switching
VPN	Virtual Private Network
FEC	Forwarding Equivalence Classes
LER	Label Edge Router
LDP	Label Distribution Protocol
RSVP-TE	Resource Reservation Protocol - Traffic Engineering
LSR	Label Switch Router
MPLS-TE	Multiprotocol Label Switching-traffic Engineering
NFV	Network Function Virtualization
TED	Traffic Engineering Database
FIFO	First in first out
ICMP	Internet Control Message Protocol

MAC	Media Access Control
ARP	Address Resolution Protocol
NAT	Network address Translation
LTS	Long Term Support
VM	Virtual Machine
RAM	Random Access Memory
LDP	Label Distribution Protocol
LSR	Label switch router
TTR	Time to Live

# Chapter One

## Introduction

### 1.1 Background of the Study

Technology has changed our lifestyles, and web services, such as the internet, have extremely impacted various aspects of our lives, including business, entertainment, education, social networking, and communication. This remarkable advancement in computer technology has led to an increasing need for high speed, reliable, scalable, and rapid services.

The rapid development of technology has completely transformed our way of life, and internet-based services, like the World Wide Web, have become an essential part of our daily routine. These services provide to diverse aspects of our lives, encompassing areas such as e-commerce, relaxation, learning, socializing, and staying connected.

The scope of the Internet has considerably outshined estimates due to its rapid development. The traditional Internet will, however, become more and more challenging to create due to the limitations of hardware capabilities and network communication protocol. Furthermore, upcoming and current network applications and services are easier. Therefore, it is necessary to address the old network shortages. The traditional network mostly consists of switches, routers, and other network infrastructure, which complicates network administration. The network users have higher expectations for the quality of network services as a result of the increase in network performance [22].

Software-Defined Networking (SDN) is utilized to increase network programmability and modernize network administration. SDN represents a novel network model that enables the separation of control and data plane functionalities found in conventional networks, resulting in a more agile, adaptable, automated, and easily controlled architecture [19]. SDN networks have three layers: the infrastructure layer (Data Plane), the control layer (Control Plane), and the application layer (Management Plane). The control plane is a controller that acts as a central management entity [6].

Furthermore, the exponential growth of worldwide IP data traffic has placed considerable pressure on traditional networks. The performance of traffic flows, particularly in terms of throughput, has significantly diminished beyond a certain level of scalability, primarily due to

limitations in the existing physical topology. As a result, the significance of dynamic network management has escalated, as it enables the overall enhancement of network throughput without forcing any modifications to the physical infrastructure, ensuring effective network maintenance [15].

Software-defined networking (SDN) is a centralized network management technology that aims to minimize the network administration and policy enforcement burdens associated with traditional IP networking [4]. This approach revolves around the separation of network intelligence, which involves packet forwarding in the data plane, and centralizing it within a logical controller.

The Open Flow for communication between separated data plane and control plain and Linux based operating system to build mininet are deployed. However, it's important to note that the OpenFlow protocol is just one example of a standard protocol used for this purpose, and there are other protocols as well [11], load balancing decisions. Load balancing either statically or dynamically [2].

There are numerous SDN Controllers, namely POX, Ryu, Trema, Open daylight (ODL), and Floodlight, etc. that were developed. Ryu follows the OpenFlow standard protocol, which allows it to interact with OpenFlow-enabled switches and devices. It supports various network applications and services, enabling developers to create custom network applications that can run on top of the Ryu controller. This flexibility makes Ryu a popular choice among researchers, network administrators, and developers in the SDN community.

One of the key features of the Ryu controller is its ease of use and extensibility. It provides a comprehensive set of APIs and libraries that simplify the development of SDN applications. The Ryu controller uses Open Flow for traffic flow arrangement through different data plane devices [21].

## **1.2 Statement of the Problem**

An advanced and radical approach to network administration is the concept of Software-Defined Networking (SDN) architecture. To deliver effective and reliable service to those who need it, the daily increasing load on the servers carried on by increased demand must be balanced [23].

The performance of software defined networks is still being questioned, and it requires the hard work of researchers; it is one of the hot topics to be investigated. The problems of Static packet batching in RYU controller refers to a process where packets are grouped together into batches based on fixed criteria. This batching process is considered more complex. In static packet batching, packets are organized into fixed-size groups or according to specific rules, without considering the network conditions or traffic similarities in real-time [21]. However, during static batching, when new arrivals, such as incoming packets or tasks, are added to the switch, it increases the workload or processing load on the switch. To efficiently handle this workload, a batch processing approach is used, where a certain number of tasks or packets are processed together as a batch.

If there is a large number of workload waiting to be processed, the schedule will allocate more time for processing in order to accommodate the higher workload. This indicates high latency mode, which means that many packets will have to wait for access to the resource. In SDN (Software-Defined Networking), switches do not directly process incoming packets. Instead, they examine the incoming packets to find a match in their forwarding table. If a match is found, the switch can make a forwarding decision based on the table entry. If no match is found in the forwarding table, the packet is routed to the controller for further processing. In this context, the controller acts as the operating system of the SDN, responsible for making decisions about packet forwarding or dropping.

Several researchers have proposed many techniques to improve SDN performance by describing their target areas, such as: According to [9] has worked on Performance Evaluation of Ryu Controller in Software Defined Networks to evaluate the performance of the Ryu controller in terms of latency and throughput. Cbench is used for measuring throughput and latency of this controller but this work is different from this research because to increase the performance of the Ryu controller in this work used MPLS on data plane on the top of Cbench.

According to [6] in this research, a comparative evaluation of the performance of popular SDN controllers has evaluated the performance of popular open source SDN (Software-Defined Networking) controllers, including ONOS, Ryu, Floodlight, and Open Daylight. The evaluation focuses on measuring latency and throughput to understand the performance characteristics of these controllers. Additionally, they provided a feature-based comparison of the controllers to aid

in decision-making when selecting a controller for specific network requirements. This work attempts to enhance the performance of the Ryu controller by using MPLS (Multiprotocol Label Switching) on the data plane, in addition to use cbench algorithms to evaluating, comparing, and optimizing the performance of SDN controllers by generating controlled simulated traffic for difficult testing and analysis.

Hence, this work attempts to answer the following research questions:

- How to enhance throughput of the network in Ryu controller SDN?
- How to decrease latency of the network in Ryu controller SDN?

### **1.3 Objectives of the Study**

#### **1.3.1 General Objective**

The general objective of this research is to enhance throughput and latency of the RYU controller based software-defined networking using MPLS.

#### **1.3.2 Specific Objectives**

To meet the general objective of the study, the following specific objectives have been identified.

- To investigate the effect of throughput and latency modes with MPLS network and without MPLS network (normal SDN).
- To implement MPLS network functionality into the RYU controller.
- To compare the performance of Ryu controller with and without MPLS network.

### **1.4 Scope of the Study**

The purpose of this study is to analyze network performance metrics throughput performance tests and latency performance tests on Ryu controller SDN using MPLS. Since it is beyond the scope of the research work resource utilization, fault tolerance, connectivity, and bandwidth usage would not be included or addressed as part of this research.

### **1.5 Significance of the Study**

This research helps various network service providers and load balancing companies to achieve their goal of updating their data centers. This paper should be important in several positions: For instance;



- Network researchers, Data science and Researchers.
- Network engineers, and Network administrators who are interested in other areas of the field, it is a good resource to know the level of software and the gaps, and the future.

Later work; opens the way for further investigation and improvement using a variety of approaches. This study technique has a significant impact on the performance of software-defined networks, and significantly improves throughput usage across the network and solves performance problems by increasing performance issues.

## **1.6 Organization of the Thesis**

The rest part of this thesis is organized as follows: Chapter two presents the literature review of the SDN architectures, presents open flow protocols, different load balancing mechanism using the data center on MPLS and related works on SDN. Chapter three presents the proposed methods of the Ryu controller using MPLS network, and research methodology. Chapter four presents implementation, result and analysis of the research, and topology of the general network. Finally, conclusion and future works is discussed in chapter five.

## Chapter Two

### Literature Review

#### 2.1 Introduction

Software-Defined Networking (SDN) introduces a model shift from traditional networks, bringing forth a range of challenges that need to be addressed, including data forwarding, load balancing, and energy management. In SDN, flow tables play a central role, with each entry containing fields such as the header, counter, and action. Upon packet arrival at a switch, a lookup operation is performed to match the flow entries. Network flow headers and counters are updated when modifications occur, such as load balancing or re-routing. By leveraging preset rules and header information, the switch processes the data flow, effectively controlling network traffic. The flow control mechanism relies on algorithms employed for load balancing, as dynamic load balancing holds significant importance in SDN's centralized controller [30].

A number of SDN load-balancing solutions have recently been introduced, with a focus on three specific aspects: data plane, control plane, and application plane load-balancing techniques [9].

#### 2.2 Overview of SDN Architecture

SDN has significantly change how to build networks, the way managing the networks, and the way running our networks. A centralized controller in SDN enables independent management of the network's data plane, resulting in reduced complexity compared to traditional networks. SDN has mainly three parts these are [32].

- **The control plane:** A software-defined network's control plane is its central component, where controllers select where to forward packets. To decide whether to forward or drop the packet, it applies a set of flow rules. To guarantee the successful transmission of the data throughout the network, the controllers are charge of determining the path between data-transmitting nodes. The controllers consult the flow tables and/or group tables of openswitches to determine the path. In conventional networks, the control plane's is to take routing decisions or to choose the optimum path for traffic routing.
- **Data plane/Forwarding plane:** Is a component of the software-defined network where actual user data transmission takes place. The set of components that make up the data plane, also known as the forwarding plane, are frequently referred to as switches. The

forwarding plane is in charge of carrying out the decisions made by the controllers, such as packet forwarding. Consequently, the data plane has a number of pathways for sending packets. The controller creates paths using group tables or flow tables. Multiple nodes are connected in traditional networks using conventional routers and switches.

- **Application plane:** By connecting with the controllers using an API known as the northbound interface, applications can be developed and installed /deployed on top of the controllers to manipulate network activities. These applications are frequently user-specific, multipurpose, and open source, and they operate without consideration for physical network infrastructure. Through the SDN's northbound interface, applications communicate with controllers to carry out certain tasks. Through northbound interfaces, the programs frequently access network resources to carry out particular tasks. The following below figure illustrates the components/general architecture of software-defined networks.

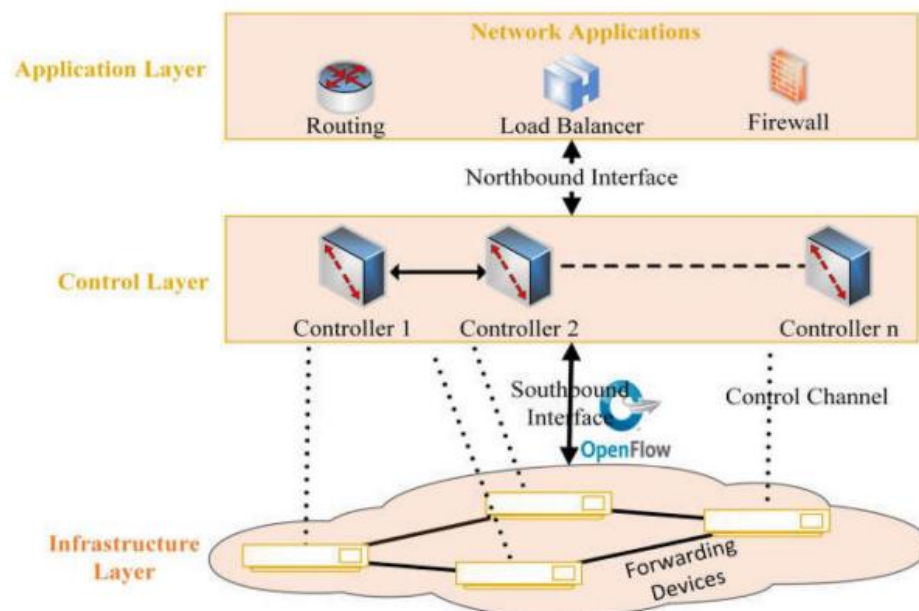


Figure 2.1: The three layers in SDN architecture

Unlike the southbound interface, such as Open Flow, there is no commonly used communication protocol in the northbound interface [13] so, each vendor should have an own protocol to set up and simplify the communication of applications with the control plane. These applications could not be interoperable, which could lead to a complicated problem. Using a southbound API, the control plane and data plane are able to communicate with one another [9].

### **2.2.1 SDN Northbound Interface**

The northbound interface is an API that enables software to be placed at the top of the controller, or what refer to as the application plane, and it allows that software to function without regard for the unique characteristics and situations of the network devices themselves [26]. This level of abstraction has several significant benefits, one of which is that network applications can operate the network service independently of the underlying physical network. The services enable the hosting of devices in a manner that allows the hosts to remain unaware that the network resources they utilize are virtual and not the original physical resources they were designed. There is no currently accepted standard for the interaction of the controller with applications. So, each brand of controller could have unique methods to ensure communication with applications. It is very difficult to manage applications having conflicting functions even though controllers provide very low-level abstraction so, there is a need for a high-level programming language that translates high-level strategies into low-level ones [13].

### **2.2.2 SDN Southbound Interface**

The southbound interface facilitates communication between the control plane and the data plane. It is the protocol used for communication between controllers and data plane/open vswitches devices. Southbound interfaces come in a variety of brands, including OVSDB, open Flow, and others, although OpenFlow is the most popular [25] [20]. Commonly used openvswitches are data plane devices that connect with the control plane using the openFlow protocol, and the Ryu controllers act on behalf of flow entries that are stored in the openvswitches flow tables. One or more controllers may be present on the control plane, and they may be on-time configure devices, establish paths, or keep an eye on the entire network from a distance. Business apps operate specific tasks at the top of the control plane by interacting with the controllers via the northbound interface. There are four ideologies of SDN networks such as:

1. **The networking and IP routing flexible** – Instead of spending many days performing manual routing to enable reachability, SDN does this in a better way, ultimately saving a lot of time. SDN flexibility enable packets or traffic to reach their destination. It does this with the help of software and dynamic algorithm with full flexibility and agility

2. **Decoupling control and data plane-** In traditional networks, the router serves as both the brain and the data forwarding layer. As a component of SDN, the centralized controller determines traffic routing. Only the payload is transmitted over the data plane to the actual destination.
3. **Offloading intelligence to a centralized controller and obtaining a centralized view of resources-** SDN offers a centralized view of substantially more effective resource allocation and continues to monitor network services
4. **Software-defined network, centrally administered, Flexible for varied demands-** Network control becomes directly programmable with centralized control planes, and applications and network services are shielded from the underlying infrastructure. Network operators can enable a variety of applications thanks to SDN, including dynamic bandwidth provisioning, automatic scale-out and scale-in etc.

### 2.3 Ryu Architecture

The Ryu SDN controller presents an architecture based on components, serving as a framework. In addition to the OpenFlow protocol, it also provides compatibility with Netconf, OF-config etc. [18]. SDN Controller architecture encompasses three layers. The uppermost layer encompasses business and network logic applications, referred to as the application layer. The middle layer comprises network services, known as the control layer or SDN framework. The lowest layer encompasses physical and virtual devices, constituting the infrastructure layer. The middle layer hosts both northbound APIs and southbound APIs. The controller provides open northbound APIs accessible to applications, including Restful administration.

RYU works OpenFlow to establish communication with the forwarding plane, containing switches and routers, aiming to alter how traffic flows are managed. Various OpenFlow switches, including OpenvSwitch and products from Centec, Hewlett Packard, IBM, and NEC, have been tested and certified to work with it [18].

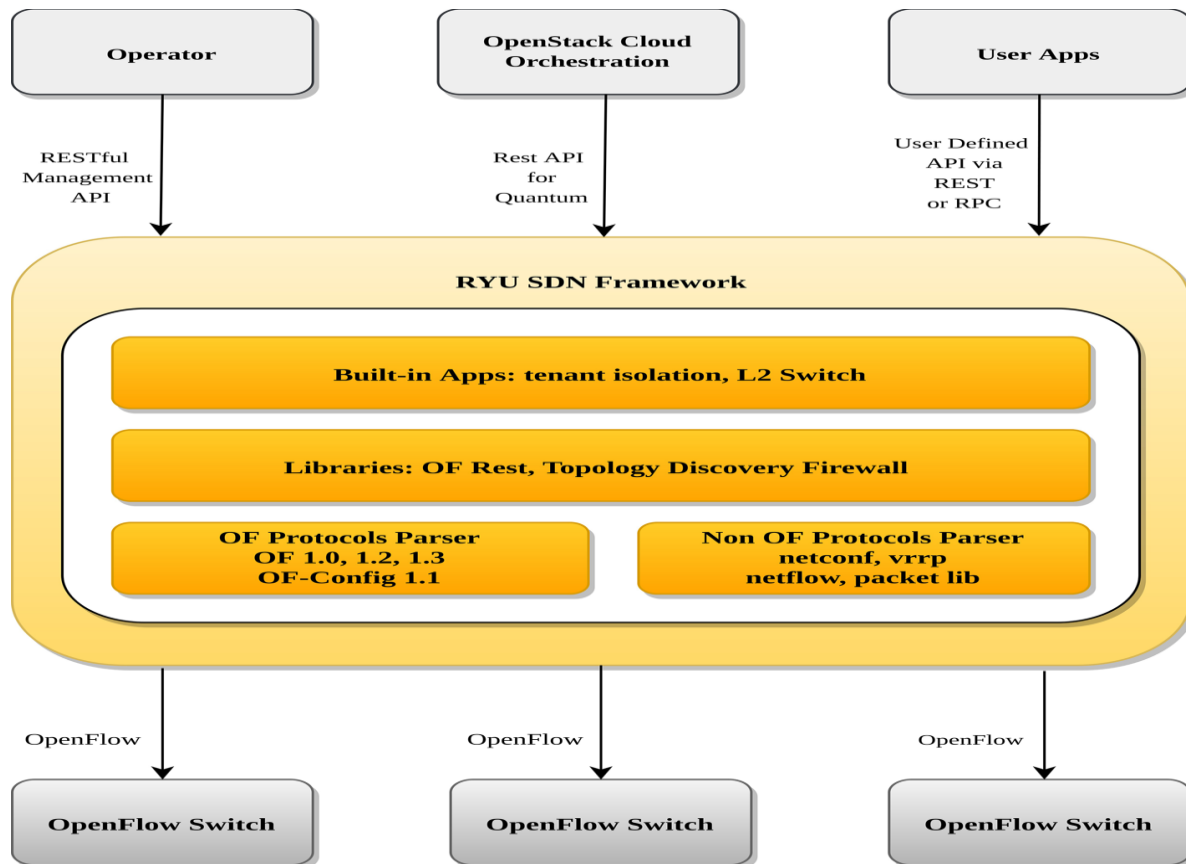


Figure 2.2 : RYU SDN controller architecture

### 2.3.1 Traditional Network Architecture

According to the traditional network architecture, a network is made to facilitate communication between end hosts and a network node's combined control and data planes. The control plane is responsible for configuring the network nodes and programming the data flow routes. It is the component of a network architecture that handles tasks related to network management. The control plane uses protocols and algorithms to establish and maintain the overall network structure and behavior. Once the paths have been determined and configured by the control plane, the control information is "pushed down" to the data plane. "Pushed down" means that the control information is communicated or transmitted to the data plane, which consists of the network devices' hardware and software responsible for data forwarding.

The control information received by the data plane includes instructions on how to handle and forward data packets based on the established paths. The data plane then performs data

forwarding at the hardware level according to this control information. This involves the actual movement of data packets through the network, ensuring they are directed along the intended routes as defined by the control plane. Because there is no control plane abstraction of the overall infrastructure, the traditional network system uses a distributed way to administer the network. As a result, networks are now difficult to configure and administer when something goes wrong. The traditional networks system's drawbacks include manual configuration, debugging, security, scalability, and mobility [18].

As below Figure 2.3 shows, the traditional network pairs data plane and control plane, making it challenging to mechanize the network. Traditional network infrastructures use physical devices or equipment to implement networking and network devices such as switches, routers, firewalls, and intrusion prevention systems, etc. Network topology determines networking flows, and each network device locally decides how to best move a packet to its destination. But as virtualized servers and network architectures based in the cloud have expanded, the capacity to rapidly install new applications without requiring significant network upgrades has become a prerequisite. The needs of today's businesses, carriers, and end users could be satisfied by these kinds of network topologies. A new network design called Software Defined Networking (SDN) decouples network control from the network infrastructure, modernizing networking architecture [15].

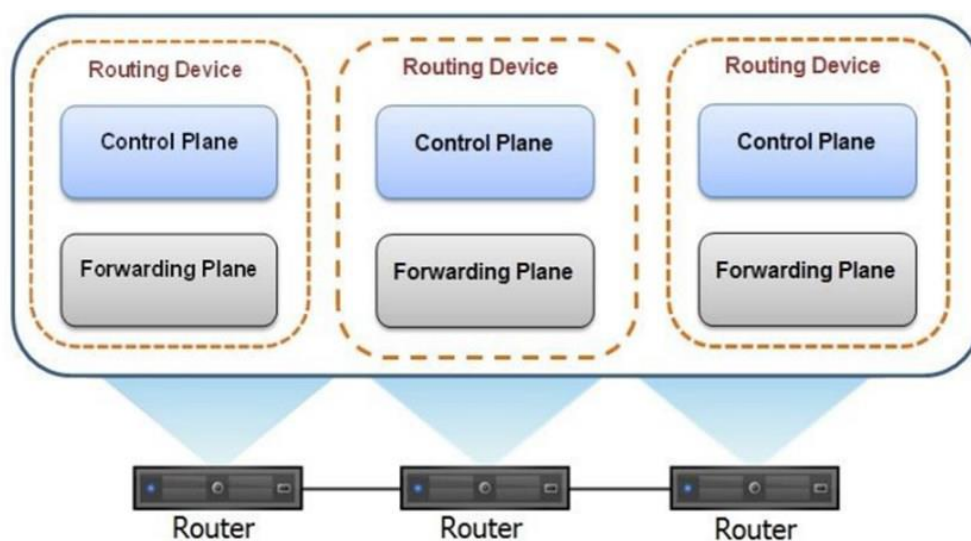


Figure 2.3 :.Legacy network architecture

### 2.3.2 Limitations of Traditional Networks

Traditional network architecture makes it impossible to satisfy current market demands. The traditional network was not designed in a manner that aligns with the current needs and expectations of end-users, service providers, and enterprises some of the drawbacks are below [15].

- **Management Complexity:** The computer network technologies are historically relied on a suite of routing protocols methodically designed to establish dependable connections between hosts spanning vast distances, ensuring high speeds across diverse network topologies. However, to address modern industry demands such as enhanced availability, security, and expanded connectivity, protocols have experienced various design approaches resulting in different separation, where each protocol provides to solving specific problems without implementing main concept abstractions. Unfortunately, this approach has led to a prominent difficulty encountered by network administrators

Network administrators often maintain a static network design to minimize or prevent service downtime resulting from changes. However, this static nature restricts the dynamic capabilities of server virtualization and subsequently increases the number of hosts requiring connectivity.

Before the advent of virtualization services, a single server would establish connections with specific clients. Today, virtualization enables applications to be distributed across multiple virtual machines, allowing for increased scalability and flexibility. Additionally, virtual machines often need to migrate to achieve balanced workloads

1. **Policy configuration challenges:** In order to maintain an enterprise network policy, network managers may be necessary to setup a large number of routers and switches that can support the network. When using virtualization, it typically takes up to hours or if not days to add a virtual machine to the network since the network administrator needs to configure and adjust Access Lists (ACLs) for the overall infrastructure.
2. **Rapid growth demands:** Due to the rapid growth requirements of data centers, the network must expand at an equivalent bound , and **Device vendor dependability**



## 2.4 Open Flow Protocol

The foundation of the whole SDN topology is the open Flow protocol.

OpenFlow's flexibility allows researchers to achieve great performance with little financial outlay and the ability to handle manufacturers' requirements for closed platforms and as well as the ability to handle manufacturers' requirements for closed platforms and to separate experimental data from actual data [1].

The interaction between the controller and the OpenFlow channel is facilitated through the utilization of the OpenFlow protocol. The OpenFlow protocols leverages to configure flows and amass network traffic information from switches. An essential element of an OpenFlow switch is the flow table, which encompasses specific instructions for each flow entry and provides guidance to the switch on how to manage incoming flows. The Flow table also connects the switch to a remote controller over a secure channel. The switch and controller can exchange packets and commands due to a flow table [10].

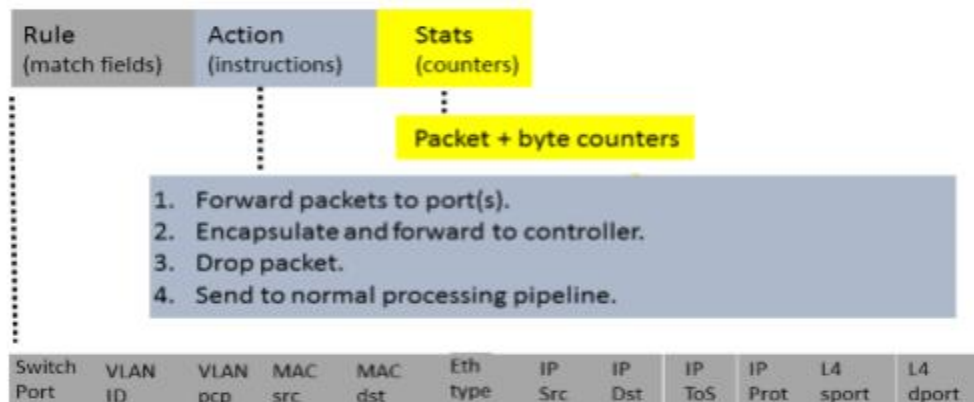


Figure 2.4 : OpenFlow table entries

A common interface for configuring the data plane switches is made up of OpenFlow protocol. Open Flow architecture is composed of three fundamental parts. The data plane is housed within OpenFlow switches, whereas the control plane consists of OpenFlow controllers. The switches and controllers are interconnected through secure control channels, forming the connection between the switches and the control plane [16].

## OpenFlow switching

OpenFlow switch specification

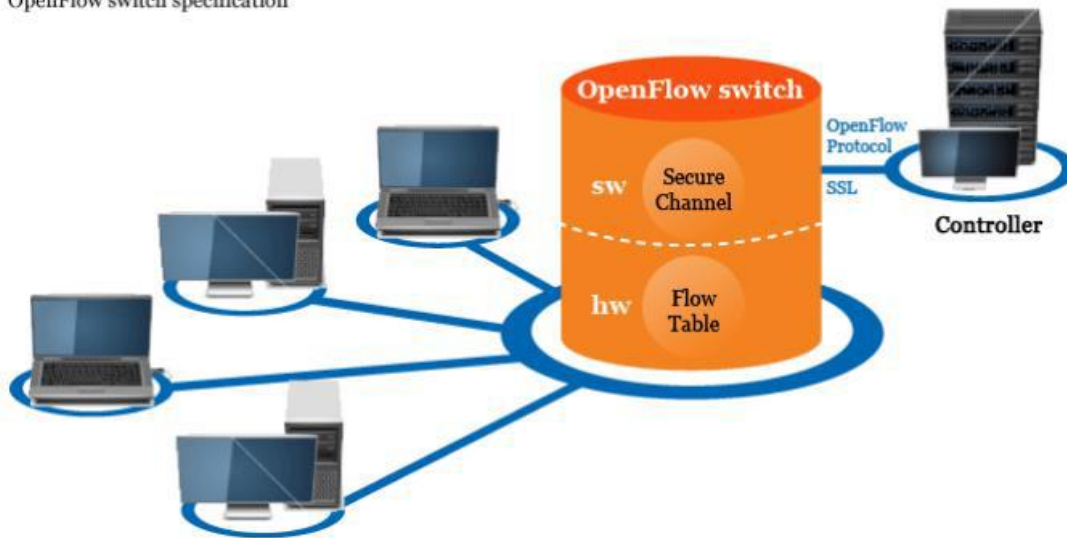


Figure 2.5: Open Flow architecture

An OpenFlow switch, functioning as a data plane device, forwards packets based on its flow table that encompasses a set of OpenFlow entries comprising match fields, counters, and instructions. These entries, known as flow rules, govern the handling of packets within the switch. Match fields in a flow table entry define the characteristics used to identify relevant packets. Counters in the flow table primarily collect statistics on flows, including packet count, byte count, and flow duration. A flow table entry header fields are responsible for evaluating various protocols according to the OpenFlow definition. Finally, OpenFlow actions, such as forwarding, dropping, and modifying field values, dictate the processing of packets belonging to specific flows [5].

The controller, a fundamental component of OpenFlow, assumes the crucial role of overseeing and regulating the flow tables within the switches. The controller is accountable for performing various operations, including the addition, alteration, and deletion of flow entries [10]. The controller also has the ability to adapt the forwarding conduct of switches. Furthermore, OpenFlow establishes a particular communication protocol that empowers the controller to command the switches via a secure control channel.

## 2.5 Packet Processing Mechanism

An OpenFlow switch consists of one or more flow tables, each with unique arrangements of fields, such as match fields, action fields, and counters. The switch is responsible for processing packets once each one has been compared to its corresponding flow table. The action for that entry is applied to the packet if a flow entry and a packet header match. This suggests that sending a packet to a certain port may be part of the action. If no match is identified, transmit a packet IN message to the controller or send it through a secure communication channel [3]. The counters are reserved for gathering flow-related data. They keep track of the number of packets, bytes, and flow times that were received. A switch analyzes the header field of each packet it receives and matches with the flow table's rules

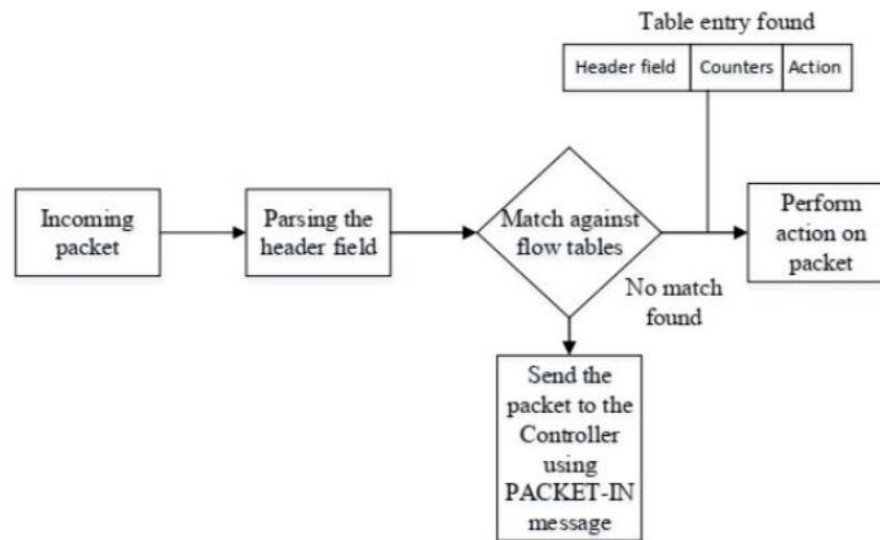


Figure 2.6: Packet flow in Open flow switch

## 2.6 Taxonomy of Load Balancing Approaches

**Load balancing** also referred to as server farm is the process of effectively distributing incoming network traffic among a collection of backend servers.

Today's high traffic websites must quickly and reliably respond to hundreds of thousands, of concurrent user or client requests for the appropriate text, photos, videos, or application data. Modern computing best practice typically requires deploying more servers in order to cost-effectively scale to handle those high loads.

These are some commonly used load-balancing methodologies in distributed computing systems. Each methodology has its advantages and trade-offs, and the selection depends on factors such as

system architecture, workload characteristics, and performance requirements. In order to improve resource efficiency and request response times, load balancing reassigns the entire load to each individual server in the cluster. This method also removes the problem where Addressing load imbalance is crucial to ensure optimal resource utilization and prevent overloaded servers from experiencing performance degradation or failure. Load-balancing algorithms and techniques are employed to distribute the workload evenly and alleviate the load imbalance. These techniques aim to dynamically adjust the workload distribution based on factors such as server capacity, current load, and system performance metrics. By mitigating load imbalance, organizations can achieve improved system performance, enhanced scalability, and better utilization of available resources.

To effectively load balance client requests among server pools, a variety of techniques can be utilized. The algorithm selected will depend on the kind of service or application being provided, the network and service state at the time of requests, and the type of service or application. Which approach is employed frequently depends on the volume of requests currently being processed by the load balancers. These benefits demonstrate the importance of server load balancing in achieving optimal performance, high availability, and efficient resource utilization in distributed systems. Various load balancing algorithms, such as round robin, weighted round robin, least load, ratio, priority, and predictive algorithms, contribute to realizing these advantages

- **High availability:** Server load balancing helps increase the overall availability of the system by distributing incoming traffic across multiple nodes. If one node experiences issues or becomes overwhelmed, other nodes can handle the traffic, ensuring that the application or service remains accessible.
- **Fault tolerance:** Load balancing enhances the fault tolerance of a system by providing redundancy. If a server fails or becomes overloaded, load balancers can automatically redirect traffic to other available servers, minimizing service disruptions.
- **Resilience:** Load balancing contributes to system resilience by distributing the workload among multiple servers. This allows the system to withstand sudden increases in traffic or unexpected spikes without being overwhelmed.

- **Scalability:** Load balancing enables the system to handle increased traffic by distributing it across multiple servers. This scalability helps ensure that the application or service can accommodate growing user demands without compromising performance or availability [42].

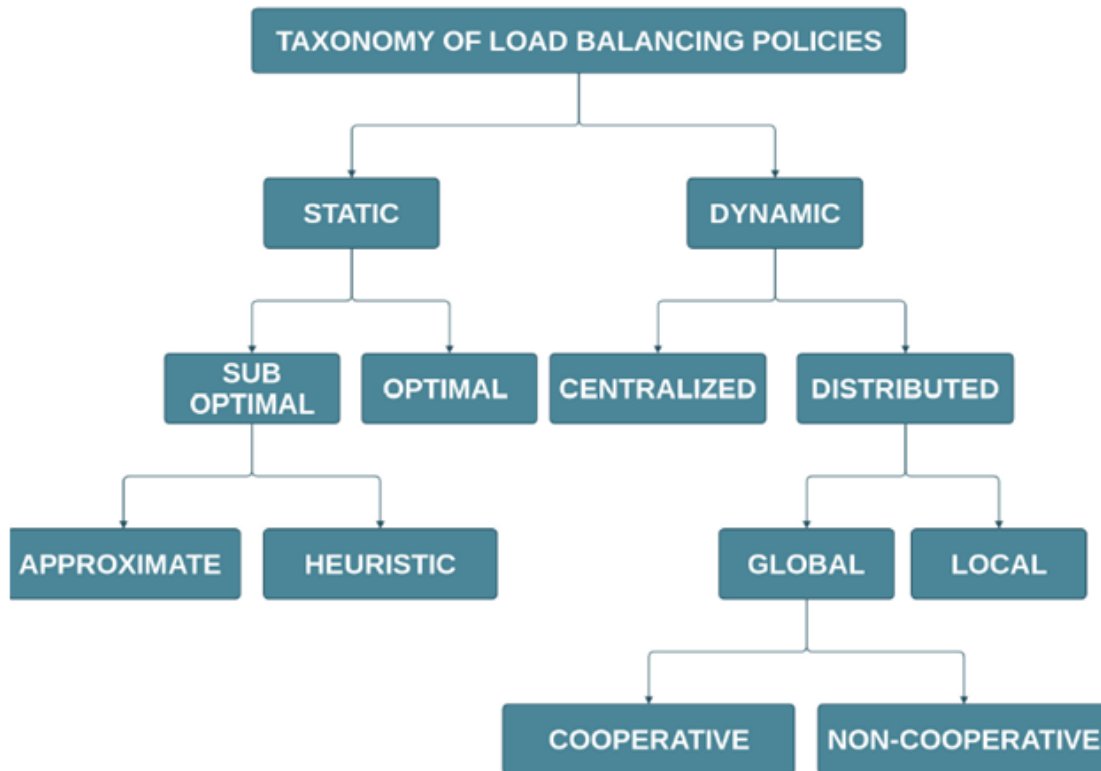


Figure 2.7: Taxonomy of Load Balancing Approaches

## 2.7 Load balancing in SDN

Software-defined networks (SDN) framework provides numerous benefits over traditional networks, including the effectiveness and convenience of network management and the application of security standards. The limitations of traditional networks, such as their capacity to provide end-to-end quality of service and efficient load balancing, are overcome by SDN.

With increasing access and data traffic, the network's processing power should rise accordingly. It is essentially expensive and a waste of resources to attempt to solve this issue by upgrading or replacing the current hardware. With the help of load balancing, the server's processing power can be increased while the time it takes to respond to user requests is decreased by distributing a large volume of concurrent admission or data movement among numerous computing devices.

**Improving QoS metrics:** The main goal of load balancing in guaranteeing complete Quality of Service (QoS) for SDN networks and improving system reliability and overall performance through the prevention of excessive device delay, performance optimization, and response time reduction, the QoS strives to improve user experience.

1. **Optimize the use of resources:** This is one of the main objectives of load balancing since efficient use of resource is essential for the effectiveness of the SDN load balancing architecture. As a result, there is a limitation to how much network resources like links, bandwidth, processors, and memory can be used. The most effective use of resources for load balancing is ensured by an appropriate resource provision algorithm.
2. **Reduce transmission latency:** The amount of time it takes the host switch to transmit data is referred to as transmission latency. This is dependent on a number of variables, including the switch's efficiency, the size of the data packets, and whether or not the transmission queue is backed up. Transmission latency serves as both a sign of network congestion and, in a sense, a measure of switch load. As a result, the SDN controller must record both the number of bytes delivered within a given time period and the transmission rate.
3. **Avoiding bottlenecks:** In the SDN network environment, load balancing techniques are essential to distribute the load correspondingly among several routers and controllers so that no router or controller becomes loaded. By making optimum use of the resources that are already available, proper load balancing can lower resource use. It also enforces failover, permits scaling, avoids bottlenecks, and speeds up response time.
4. **Increase throughput:** A high-performance network should have high throughput, which is only possible if the workload and resources are distributed evenly across the nodes. This is the volume of data that was successfully moved from one location to another over a protracted period of time.

## **2.8 Types of load-balancing algorithms: Static versus Dynamic**

Static load balancing techniques only take into account data on the typical system behavior. Static load balancing algorithms ignore the condition or load of the system's nodes right now. The workload distribution is decided upon at run-time using dynamic load balancing. Depending on the most recent information gathered, the master offers the worker a new assignment to do. Since the workload distribution is carried out while the program is running, performance might

well be improved. However, the improved performance comes at the expense of increased communication costs. So, the overhead associated should be in reasonable limit to achieve better performance [29].

Load-balancing is a method to distribute load among network components to enhance QoS and maximize network performance. By allocating or Load balancing techniques shift the load to support service providers and end users and algorithms significantly contribute to increased efficiency.

- **Essential for Load Balancing:** The network's servers are being overloaded as the number of concurrent requests from client's increases; as a result, the load must be balanced in order to provide better service and observe QoS standards. Neglecting this issue results in links failing and sometimes server crashes. In contrast to traditional networks, software-defined networks separate all control planes from switches and place them in a centralized unit known as a controller.
- **Significance of Load-balancing:** In SDN interfaces are used to connect the three layers. The infrastructure layer's network devices forward requests to the control plane. On the other hand, it is necessary to fulfill the demands placed on applications by various services at the application layer. Therefore, the control plane plays an essential intelligent role in satisfying the requirements. The amount of requests from clients is increasing along with customer demand for cloud services, which increases the workload on the networking components to manage them [8].

## 2.9 MPLS Architecture

In MPLS architecture, packets are assigned labels at the ingress router (also known as Label Edge Router or LER). The labels are distributed throughout the network using protocols like Label Distribution Protocol (LDP) or RSVP-TE (Resource Reservation Protocol - Traffic Engineering). Each router along the path, known as Label Switch Router (LSR), examines the packet's label and forwards it based on the label forwarding table. This label-based forwarding allows for faster and more efficient routing decisions, as the routers don't need to perform complex IP lookups for each packet. MPLS networks classify packets into Forwarding Equivalence Classes (FECs). FECs group packets with similar characteristics or forwarding requirements, such as destination IP address, QoS requirements, or application type. Each FEC is associated with a unique MPLS label. The labels represent the specific treatment and forwarding

instructions for packets within the MPLS network. By using FEC-based forwarding, MPLS architecture provides flexibility in implementing different traffic engineering and QoS policies.

Traffic Engineering and Virtual Private Networks (VPNs): MPLS architecture enables traffic engineering capabilities, allowing network operators to optimize the utilization of network resources. Traffic engineering techniques, such as MPLS Traffic Engineering (MPLS-TE), enable the establishment of explicit paths for traffic, ensuring efficient utilization of available bandwidth and avoiding congestion. Additionally, it is widely used for implementing Virtual Private Networks (VPNs). MPLS VPNs provide secure and isolated communication between different sites of an organization over a shared MPLS network. By assigning unique labels to VPN traffic, MPLS architecture ensures privacy and separation of customer traffic within the network.

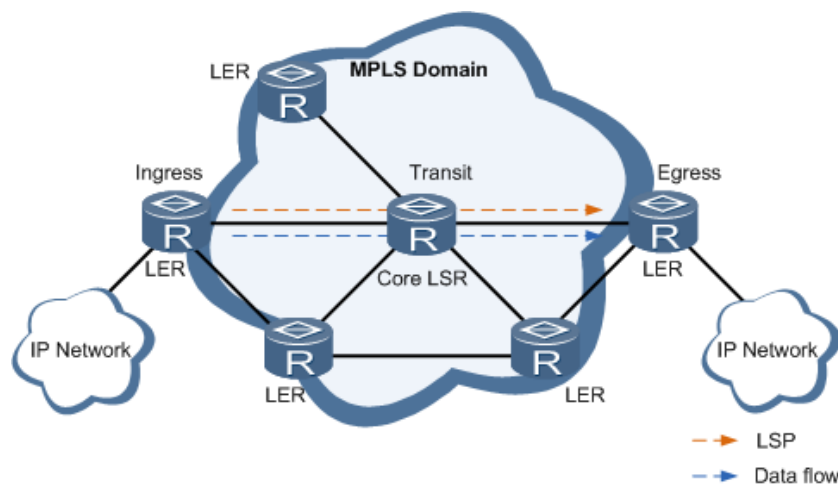


Figure 2.8: General MPLS Architecture [41]

### 2.9.1 MPLS Process using LSR

When an LSR receives an incoming packet, it examines the MPLS label carried in the packet's header. This label serves as a key to perform a lookup in the LSR's label forwarding table. The label forwarding table contains entries that map MPLS labels to the corresponding outgoing interfaces or next-hop routers. Each entry in the table specifies the label value, the corresponding outgoing interface or next-hop router, and any necessary instructions for label operations.

Based on the lookup result, the LSR determines the appropriate outgoing interface or next-hop router for the packet. It then encapsulates the packet with the appropriate MPLS label for the next hop and forwards it along the determined path. This label-based forwarding mechanism



allows LSRs to make fast and deterministic routing decisions, as they don't need to perform complex IP lookups for each packet. Instead, they rely on the MPLS label for forwarding.

In some cases, LSRs may need to perform label operations, such as swapping or stacking labels. Label swapping involves replacing the incoming MPLS label with a new label based on the label forwarding table entry. This swapping allows LSRs to adjust the label based on the network's requirements. Label stacking involves adding additional labels to the existing MPLS label stack. This stacking occurs when a packet needs to traverse multiple MPLS domains or undergo different label-based operations.

The label lookup and forwarding process continues at each LSR along the path until the packet reaches the egress LER (Label Edge Router). The egress LER determines the exit point of the packet based on the MPLS label. It removes the MPLS label and forwards the packet to the appropriate destination based on the network layer header information.

The following figure 2.8, 2.9, 2.10 illustrates how packets are labeled and forwarded in MPLS backbone.

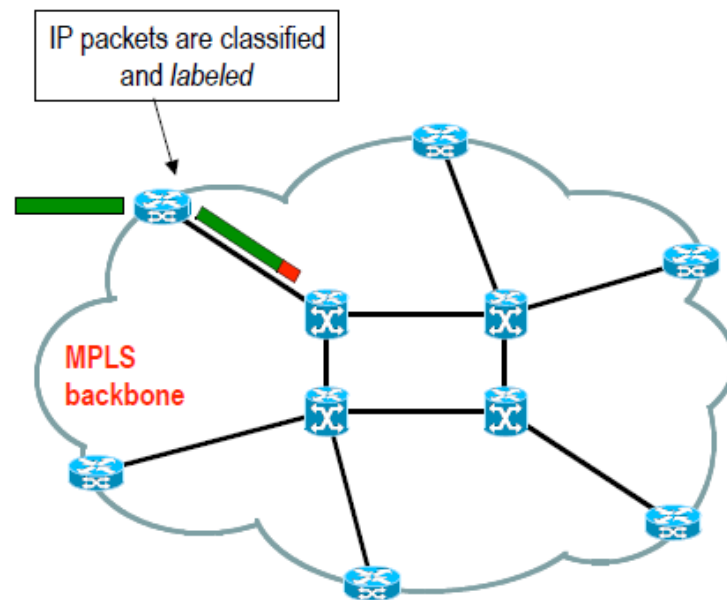


Figure 2.9: Label Insertion [41]

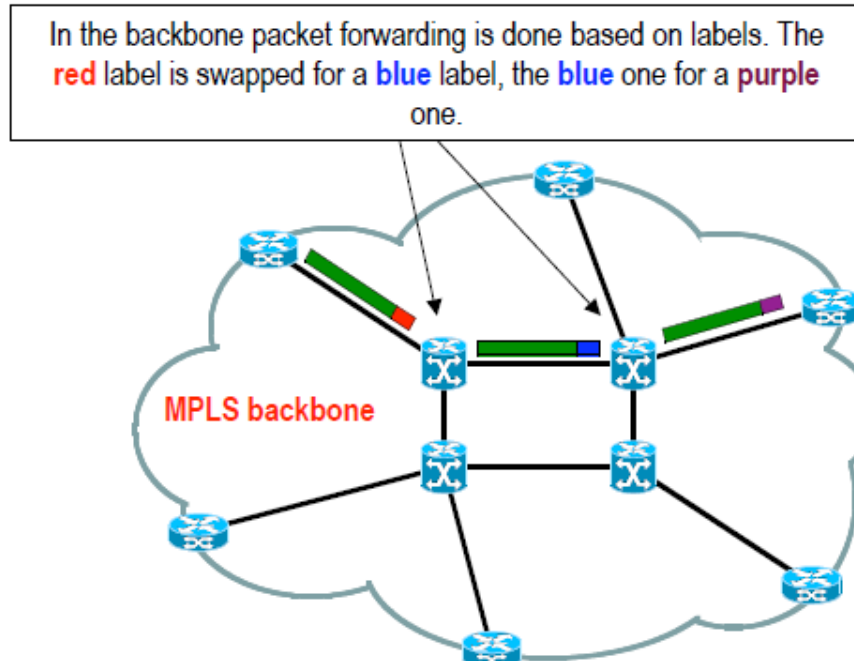


Figure 2.10: Label Swapping [41]

### 2.9.2 Benefits of MPLS networks

MPLS (Multiprotocol Label Switching) networks, when combined with SDN (Software-Defined Networking) principles, offer additional benefits that leverage the flexibility and programmability of SDN. Here are some key benefits of MPLS networks in an SDN environment:

1. **Simplified Network Management:** SDN provides centralized control and programmability, allowing network administrators to manage and configure MPLS networks more efficiently. With SDN, the network management tasks, such as provisioning MPLS tunnels, setting up MPLS labels, and traffic engineering, can be automated through a centralized controller, reducing complexity and operational overhead.
2. **Dynamic Traffic Engineering:** SDN-enabled MPLS networks can leverage the real-time traffic and network state information provided by the SDN controller to dynamically adjust traffic paths based on current network conditions. This dynamic traffic engineering capability improves network efficiency, optimizes resource utilization, and enables better load balancing and congestion avoidance.

3. **Rapid Service Deployment:** SDN allows for on-demand service provisioning and rapid deployment of MPLS-based services. By programmatically configuring MPLS tunnels and labels through the SDN controller, service providers can quickly deliver new services or make changes to existing services without manual configuration on individual network devices.
4. **Service chaining and Network Function Virtualization (NFV):** SDN combined with MPLS enables service chaining and integration with NFV. Service chaining refers to the ability to direct traffic through a series of virtualized network functions (VNFs) in a specific order. MPLS can be used to establish the necessary tunnels and labels to steer traffic through different VNFs, allowing for flexible service chaining and efficient deployment of network services.
5. **Enhanced Security and Privacy:** SDN-based MPLS networks offer enhanced security and privacy features. By leveraging the programmability of SDN, network administrators can enforce security policies, implement access control, and isolate traffic flows more effectively. MPLS-based VPNs in SDN environments can provide secure and private communication between different network segments or customer sites.
6. **Seamless Integration with SDN-enabled Applications:** SDN enables the integration of applications and network services with the MPLS network. By leveraging APIs and northbound interfaces provided by the SDN controller, applications can directly interact with the MPLS network, enabling the development of innovative services and applications that can control and utilize MPLS functionalities.
7. **Scalability:** MPLS involves designing and implementing a network architecture that can accommodate growth, handle increased traffic volumes, support a large number of network elements, efficiently manage routing information, and provide seamless provisioning of services. By ensuring scalability, MPLS networks can adapt to evolving requirements, accommodate future growth, and deliver efficient and reliable connectivity and services.
8. **MPLS based traffic engineering architecture:** MPLS networks is a framework that enables network operators to optimize the utilization of network resources

and control the flow of traffic in a more efficient and controlled manner. It involves the use of specialized mechanisms and protocols to establish explicit paths for traffic, considering factors such as bandwidth, latency, and link utilization. At the core of MPLS TE architecture is the concept of constraint-based path computation, where algorithms calculate the optimal paths that satisfy specified constraints. This is facilitated by a Traffic Engineering Database (TED) that stores information about the network topology, link attributes, and available resources. Signaling protocols like RSVP-TE are utilized to establish and maintain explicit paths across the network, ensuring that network resources are efficiently allocated and quality of service requirements are met. MPLS TE also incorporates traffic splitting and load balancing techniques to distribute traffic across multiple paths, optimizing resource utilization. Additionally, fast reroute mechanisms minimize service disruptions by quickly switching traffic to backup paths in case of link or node failures, enhancing network reliability and resilience.

By combining the benefits of MPLS networks with the programmability and centralized control of SDN, organizations can achieve greater flexibility, scalability, and efficiency in their network operations. However, it's important to plan and design the SDN-enabled MPLS network carefully, considering factors such as traffic patterns, service requirements, and the capabilities of the SDN controller and network devices to fully leverage the advantages of this integration.

## **2.10 Related works**

For a comprehensive understanding of the subject and to provide effective solutions to the identified problems, extensive research has been conducted by reviewing various sources. These sources include published articles in journals, reference books, conference papers, and other online resources. The purpose of this review is to gain insights into the concept of software-defined networking (SDN) and its current state-of-the-art. The study extensively examines previously published articles and scholarly works written by researchers and experts in the field. By analyzing these research articles and literature, a thorough understanding of SDN and its various aspects is achieved. The review covers topics related to the performance enhancement of the RYU controller in the context of SDN. By exploring into the existing body of knowledge, the study aims to gather valuable information and insights that can contribute to addressing the identified challenges and improving the performance of the RYU controller. The review of

related works provides a foundation for the study, enabling the researchers to build upon the existing research and identify potential gaps or areas that require further investigation.

MPLS introduces label-based packet forwarding, where packets are assigned labels and forwarded based on these labels instead of performing complex IP lookups. This label-based forwarding mechanism significantly improves packet forwarding efficiency, reducing the processing overhead on routers and enhancing overall network performance. By streamlining the forwarding process, MPLS enables faster and more efficient packet delivery, resulting in improved network responsiveness and reduced latency. In the context of traditional networks, a recent study conducted [35] focused on enhancing the quality of service (QoS) through the implementation of segment routing multiprotocol label switching (SR-MPLS). This research work in the area of traditional networks explored the utilization of SR-MPLS as a means to improve QoS. The findings of the study demonstrated significant reductions in packet loss and jitter, indicating the effectiveness of the proposed approach. Furthermore, the study identified the importance of considering the influence of SR-MPLS on resource utilization, suggesting that optimizing resource allocation can further enhance network performance and QoS.

According to [36] mentioned study proposed a solution to improve resource allocation and study the interdependency between flows by utilizing both Software-Defined Networking (SDN) and Multi-Protocol Label Switching (MPLS) technologies.

They introduced two resource re-allocator modules: the flow level resource re-allocator and the LSP level resource re-allocator. These modules aim to optimize resource allocation and prevent congestion in the network. The flow level resource re-allocator is responsible for managing individual flows. It uses openflow switches to assign flows to existing MPLS network Label Switched Paths (LSPs). When there is a risk of congestion due to overflow in a link, the flow level resource re-allocator re-routes the flow to avoid congestion, based on a predefined threshold. However, if the flow level resource re-allocator is unable to control congestion, the LSP level resource re-allocator comes into play.

The proposed approach aims to improve resource utilization and overall throughput in the network. However, the authors did not take into account the issue of packet loss during rerouting. This means that there is a potential for packet loss when flows are re-routed to avoid congestion

and propagation delay of every link in the network is the same. The propagation delays can vary depending on the physical characteristics of the links and the distance between nodes. Therefore, considering all links to have the same propagation delay may not accurately represent the network's behavior.

According to [36] the research approach, the authors combine Software-Defined Networking (SDN) and Multiprotocol Label Switching (MPLS) to enhance resource allocation and investigate flow interdependency. They introduce the "flow level resource re-allocator" as a key component. This approach involves the use of OpenFlow switches to direct flows onto the existing MPLS network Label Switched Paths (LSPs). The "flow level resource re-allocator" is responsible for preventing congestion by dynamically rerouting flows when a link's utilization exceeds a predefined threshold. This measure ensures efficient load balancing and prevents individual links from becoming overloaded, ultimately leading to improved resource utilization and higher throughput. By leveraging the advantages of both SDN and MPLS, their approach explores how these technologies can optimize resource allocation in the network and study the relationships between flows. Through the flow level resource re-allocator, they demonstrate how SDN-based dynamic rerouting can complement the capabilities of MPLS, leading to a more efficient and resilient network.

According to [6] this study, the core element of Software-Defined Networking (SDN) network, the controller, is explored. With SDN being an alternative to traditional networks, various controllers have been developed, including Beacon, Floodlight, RYU, OpenDaylight, ONOS, NOX, and POX. Due to the diversity of SDN applications and the availability of different controllers, choosing the most suitable controller has become an application-dependent process. To address this, the study evaluates different SDN controllers based on their impact on SDN Quality of Service (QoS) performance.

The evaluation focuses on comparing the performance of POX and RYU controllers using the Mininet and Miniedit emulation tools. For the emulation, Mininet, iperf3, ping, and the POX and RYU controllers were run on an emulation machine with specific specifications. The study investigates the QoS performance in terms of key parameters, namely Throughput, Round-Trip Time (RTT), and Jitter. These parameters are measured under different traffic scenarios involving TCP, UDP, and ICMP traffic. By conducting evaluations for both POX and RYU

controllers, the study aims to gain insights into how these controllers impact QoS performance in SDN networks. Through this investigation, the study can provide valuable information to help users and network administrators make informed decisions when choosing the most suitable controller for their specific SDN application and use case.

According to [33] furthermore, a single-threaded centralized controller can still outperform multi-threaded controllers in simplified topologies, but multi-threaded controllers are preferred for complicated conditions. Second, the physical topology of the controller has a direct impact on various performance characteristics. This work has not gone in this path; however, topology-specific controller placement experiments, particularly for specialized networks, might be interesting future work. Third, quantifying the performance of specialized network controllers is a significant difficulty.

According to [39] this researchers to address the interoperability challenge, it is essential to establish standardized interfaces for communication between the SDN control plane and non-SDN control plane components. This interface acts as a bridge, enabling effective communication and coordination between traditional networking devices and the SDN architecture. These interfaces play a vital role in facilitating the integration of SDN into existing network infrastructures, allowing for a smooth migration and coexistence of SDN with legacy networking systems.

By providing a standardized communication interface, SDN can work alongside non-SDN control plane components, supporting gradual deployment and enhancing network flexibility. Moreover, such interfaces enable the development of hybrid SDN deployments, where SDN and traditional networking coexist, facilitating the transition and adoption of SDN at a pace suitable for the organization's needs.

According [37], implemented MPLS within a Software-Defined Network (SDN) environment. The main idea behind this implementation was to allow the controller to assign labels to packets when requested by the edge switches. In this approach, both edge switches and core switches are utilized. The edge switches forward packets to the core switches using labels assigned by the controller. Additionally, the controller is responsible for installing the path on the core switches. To store the labels, the author introduced the concept of label mapping. All labels assigned to a particular packet are converted into a static label and stored in the MAC address table. The MAC

address table contains forwarding rules in the form of static Address Resolution Protocol (ARP) entries. Each entry includes the label value and the destination port number on the switch.

## 2.11 Gap Analysis from Related Works

In this section from the related works discussed above the research.

Table 2.1: Gap Analysis of Related works

Author	Title of the research	Methods	Conclusion	Gap identified
Mohammad et al [36]	SDN-based resources allocation in MPLS networks: a hybrid approach	SDN-MPLS networks	Improve the resource utilization and throughput	Observing the loss of packets when re-routing traffic.
John, B[40]	implementing MPLS with Label Switching in SDN	Enabling the controller to assign labels to the packets when the edge switches request for labels	High hit rate to fill flow rules in the entire topology	Offloads the controller
Askar, S., & Keti, F [6]	Performance Evaluation of Different SDN Controllers	SDN uses mininet and miniedit an emulated tool	Improve throughput , and jitter	Considering latency and other metrics
E.Gamess et al 33]	SDN Controllers: Benchmarking & Performance Evaluation	OpenFlow protocol, SDN controllers	They propose Open Flow Benchmarking Tools by customizing the feature of Cbench. They perform performance evaluation on OpenDaylight, Floodlight, Ryu, OpenMUL controllers.	Other than OpenFlow protocols has not been addressed.
S.Mishra et al.[39]	A Survey on Software Defined Networking with Multiple Controllers	OpenFlow protocol, SDN controllers	Survey research challenges in the SDN network, identify the best SDN controller and improve the performance.	Interoperability in SDN and non-SDN controllers and performance constraint in SDN controllers.



## Chapter Three

### The Proposed Enhanced RYU controller using MPLS

#### 3.1. Introduction

Chapter three the proposed of enhancement of Ryu controller using MPLS network. In this chapter, the research methodology, proposed methods are described. In this proposed method design the Ryu controller, being an SDN controller, controls the capabilities of the OpenFlow protocol to manage and control the behavior of network devices. By utilizing OpenFlow features, the Ryu controller can establish whole communication with both the Openvswitch and the legacy network devices. To facilitate communication between the Ryu controller and the Openvswitch, the OpenFlow protocol is active. OpenFlow acts as a conduit for the Ryu controller to send instructions and commands to the Openvswitch or Openflow event controller. These instructions encompass a range of actions, including packet forwarding, header modification, and MPLS label assignment.

The MPLS devices are configured to support, which is a best technique used in packet-switched networks. MPLS enables efficient packet routing by utilizing labels attached to packets. Through the MPLS network configuration, the context of the MPLS network, the Ryu controller is capable of setting up MPLS labels on the packets. These labels play an essential role in the label switching mechanism, which enables efficient and accurate routing. By assigning MPLS labels, the Ryu controller can direct packets through the network, ensuring optimal path selection and efficient data transmission.

#### 3.2 RYU Controller

The Ryu controller, developed and designed by NTT Company in Japan, is an open-source SDN controller. It offers a range of tools and libraries that facilitate the easy and convenient of SDN networks. The Ryu controller is compatible with various versions of the OpenFlow protocol, including 1.0, 1.2, 1.3, and 1.4. It is implemented entirely in Python, which is a widely accepted programming language used for communication between the control and forwarding layers (data plane) in the networking domain. The Ryu controller is recognized as a component-based and open-source software defined networking framework. It is developed in Python, providing a flexible and extensible platform for managing network devices. It supports multiple southbound

protocols for device management, including OpenFlow, NETCONF (Network Configuration Protocol), OFConf (OpenFlow Management and Configuration Protocol), and other protocols. Additionally, the Ryu controller also supports Nicira extensions, enhancing its capabilities and compatibility with various networking technologies and architectures.

- It is an open-source platform for managing networks at the software level, rather than the traditional hardware-level control.
- Ryu provides a well-defined Application Programming Interface (API) between the control layer and the data forwarding layer (data plane), allowing for the application of custom control logic to the network.

### **3.3 Research Methodology**

The methodology of this research, which aims to enhance the performance of the RYU controller in SDN through the utilization of the MPLS (Multiprotocol Label Switching) method and in addition to testing SDN infrastructure by emulating controlled workloads. It generates a substantial volume of OpenFlow messages to scale how efficiently controllers and switches handle diverse traffic patterns. Cbench tool allows network administrators and researchers to assess the scalability, throughput, and latency of their SDN deployments under different traffic conditions. To achieve the study's objectives and address the research questions, various aspects of the existing RYU controller are examined. To investigate the networks of the MPLS in this context, a comprehensive review of different methodologies is conducted. Relevant literature sources such as books, journals, and conference papers are extensively researched. The aim is to gather insights from the state of the art in SDN and related MPLS-based systems. These methodologies are carefully evaluated and compared, considering two performance evaluation metrics.

By adopting this research methodology, a thorough understanding of the existing RYU controller's features is gained, enabling the identification of areas for improvement. These thesis approach is then developed and implemented, facilitating performance analysis to assess its impact.

### 3.4 Cbench Algorithm

To conduct throughput and latency tests on the controller [34]. For the Ryu controller with a different number of switches (1, 2, 4, 8, 16, and 32 etc.) used to investigate the performance evaluation of the controller.

cbench is a benchmarking tool for controllers

Algorithm:

```
pretend to be n switches
create "n " openflow sessions to the controller
if latency mode (default):
    for each session:
        1) send up a packet in
        2) wait for a matching flow mod to come back
        3) repeat
        4) count how many times #1-3 happen per sec
else in throughtput mode (i.e., with '-t'):
    for each session:
        while buffer not full:
            queue packet_in's
            count flow_mod's as they come back
```

Table 3.1: Cbench Algorithm

### 3.5 Proposed Method

MPLS (Multiprotocol Label Switching) network can be used in an SDN (Software-Defined Networking) environment as part of the data plane to provide traffic engineering and path optimization. SDN separates the control plane from the data plane, allowing network administrators to manage the network more easily and efficiently. In an SDN architecture, the Ryu controller consists of a centralized controller that manages the networks. It can be used to optimize traffic flows between different network nodes by assigning labels to packets that identify the desired path through the network. By using MPLS network in the data plane, network administrators can dynamically adjust traffic routing and prioritize traffic based on

performance requirements. The Ryu controller manages the Open flow Vswitch via the Open Flow protocols. The packet initially arrives at the data plane, specifically the OpenFlow switch. The packet is initially received by the data plane, more precisely the OpenFlow switch. It helps us which path the packet either follow or not, this approach reduces unnecessary flows that offload the controller and even the path. So, packets are processed and transferred via a pre-configured path.

The packets are encapsulated whenever they are transferred from source to destination as shown in Figure 3.1. The RYU controller takes on the responsibility of coordinating the entire system. MPLS network is a technique used in networking that allows, it would be used to route traffic through the network based on labels, rather than using traditional IP routing protocols. Overall, this thesis should be scalable the controller with the help of MPLS network services.

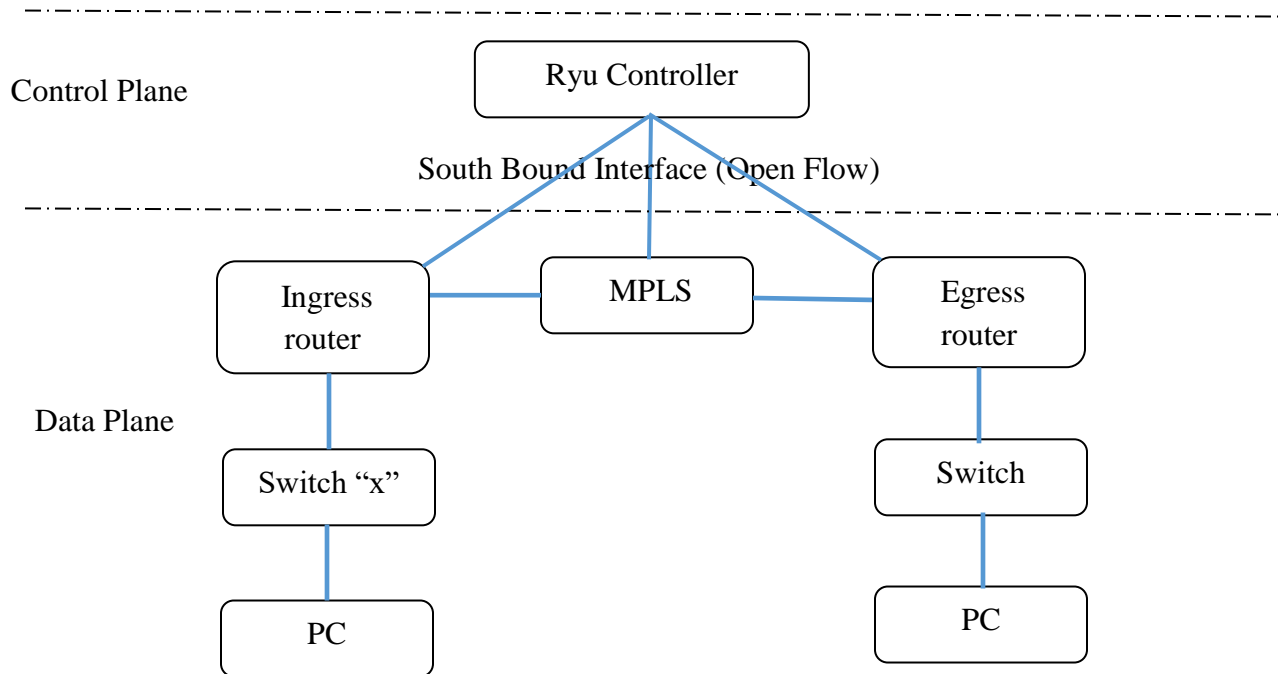


Figure 3.1: General Methods using MPLS

Multiprotocol Label Switching (MPLS) is a protocol-independent method utilized in computer networks to optimize the effectiveness and efficiency of packet forwarding. It has a mechanism for fast and scalable forwarding of IP packets, as well as packets of open flow protocols, and introduces the concept of labels, which are short identifiers assigned to packets or flows.

The internal components of OpenFlow comprise multiple flow tables and a group table, responsible for packet lookups and forwarding operations. As shown the above Figure 3.1, the data plane links with the Ryu controller via the OpenFlow protocol, is a secure communications protocol used to protect data transmitted. It ensures that data is not accessed by unauthorized users and provides encryption for data sent over the network and the Ryu controller manages all networks. They are used to define the rules for how packets should be forwarded, as well as specify which interfaces to use and which actions to take when a packet arrives at a router. All of these components are related in that they are used to manage and optimize the performance of Ryu controller.

MPLS networks are characterized as single-threaded systems, encompassing diverse functionalities. They incorporate a receive queue for events, typically adhering to a First-In-First-Out (FIFO) approach to maintain event order. Moreover, each application includes a dedicated thread responsible for processing events from the queue. The primary loop of this thread retrieves events from the receive queue and invokes the appropriate event handler. Consequently, the event handler is executed within the context of the event-processing thread, operating in a blocking manner, which means that no additional events for the Ryu network would be processed until control is relinquished.

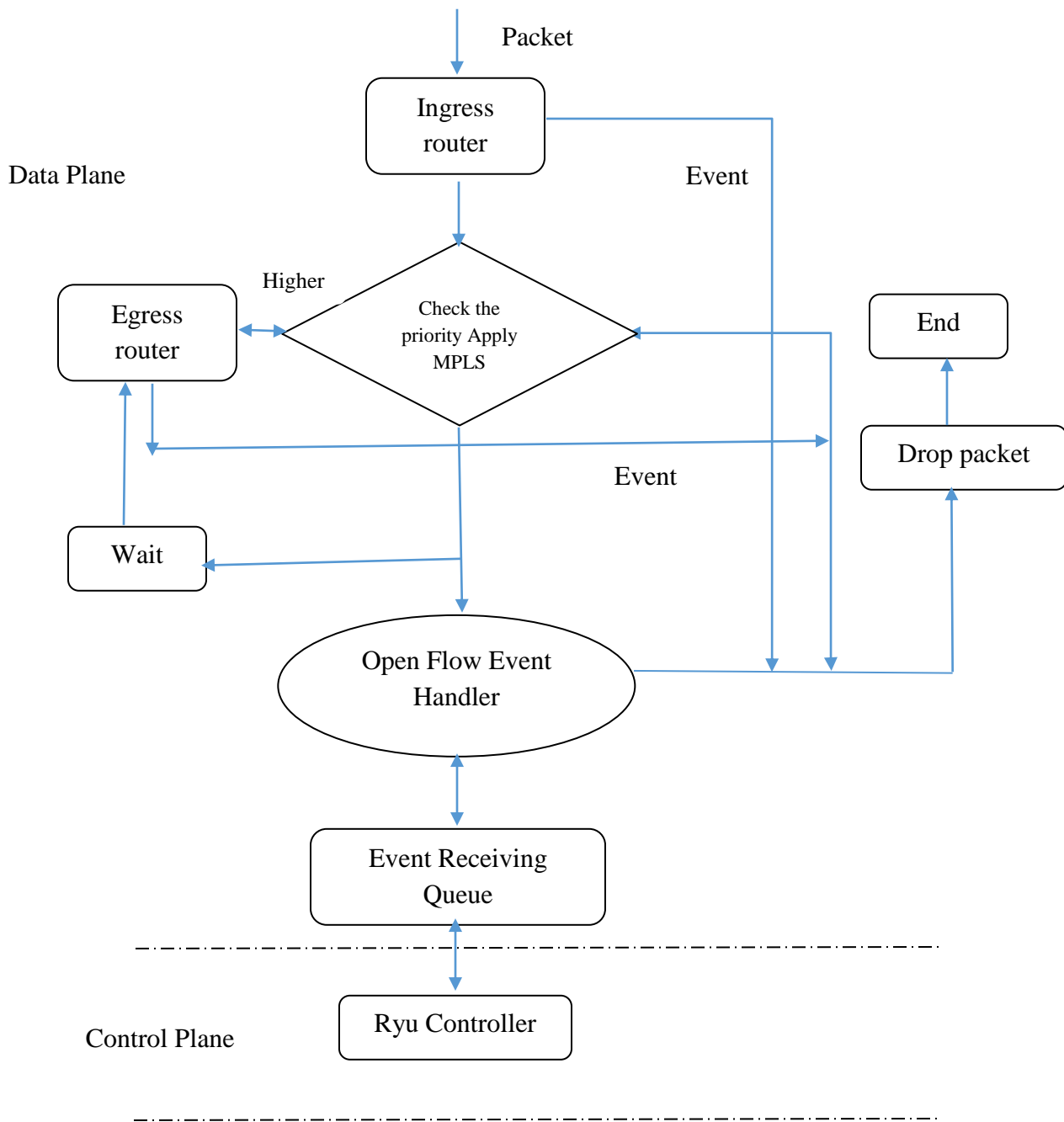


Figure 3.2 Detail Flowchart Ryu Controller with MPLS network

### 3.6 Building blocks of MPLS

Multiprotocol label switching contains the following components.

➤ **An ingress router**, also known as an entry router or edge router, is a network device located at the edge of a network. It serves as the entry point for incoming traffic from external networks into an internal network or network segment. The primary function of an ingress router is to receive packets from external networks and forward them into the internal network based on routing and forwarding decisions.

➤ **An egress router** is exit router or edge router, is a network device located at the edge of a network. It serves as the exit point for outgoing traffic from an internal network or network segment to external networks. The primary function of an egress router is to receive packets from the internal network and forward them to the appropriate external destination based on routing and forwarding decisions.

Configuring and managing the priority of MPLS within an ingress and egress router involves emphasizing the efficient flow of traffic and handling of events in the network. The prioritization process is crucial for proper event handling and control through the Ryu controller, and managing the Event Receiving Queue to ensure whole operation.

In an MPLS network, the ingress router plays a fundamental role in packet classification and assigning appropriate labels. MPLS determines the routing path based on labeled packets, the ingress router needs to have robust configuration settings that identify traffic types and apply corresponding Quality of Service (QoS) policies. These policies dictate how the router gives packets, assigning priority levels based on defined parameters

Simultaneously, the egress router is responsible for forwarding labeled packets based on the assigned labels and handling their priority according to the QoS policies defined by the ingress router. It ensures that traffic flows smoothly through the network while maintaining the prioritization set at the ingress point.

Ryu controller may command specific MPLS-enabled routers or switches to alter their label operations, re-route traffic through different paths, or prioritize certain types of MPLS traffic. These handlers receive notifications from the network elements, such as routers, regarding events like changes in MPLS label information or packet prioritization alterations. To connect the event

handlers to the routers, the Ryu controller utilizes various protocols like OpenFlow to communicate with the network devices and exchange information.

The Event Receiving Queue acts as a buffer for incoming events from the network, enabling the Ryu controller to process and respond to these events efficiently. Prioritizing this queue ensures that critical events are handled promptly and avoids potential bottlenecks in event processing



## Chapter Four

### Implementation, Result and Analysis

#### 4.1 Setup of the simulation Environment

The simulation environment has been configured using a PC:

- Processor: Intel(R) Core™ i7-7500U CPU @2.90GHZ
- Installed RAM:8.0 GB
- System type: Windows 10 64-bit Operating System
- Oracle VM Virtual box: an open source hypervisor type 2 to run Ubuntu 20.04 LTS.
- Ubuntu 20.04 LTS
- Mininet: installed inside Ubuntu 20.04 LTS
- RYU: installed Ubuntu 20.04 LTS

#### 4.2 Topology

In this research created a network topology containing MPLS core network integrated with SDN; the core network contains five MPLS enabled router having the capability to process the packets on the data planes. In this scenario, the controllers are responsible for bottom to up flows detail below figure, whereas low level flows, such as data transmission among low level users is handled by MPLS.

The GNS3 are installed on virtual machines Ubuntu 20.04 LTS, and in this work integrated them with GNS3 to operate together. Mininet is used to create topologies integrated with GNS3, then added the interfaces of routers with MPLS capability to openvswitches interfaces so as to establish the communication between the MPLS and mininet topology. The interfaces of the routers connected to openvswitches are managed by the openflow controller while they are sending and receiving the packet.

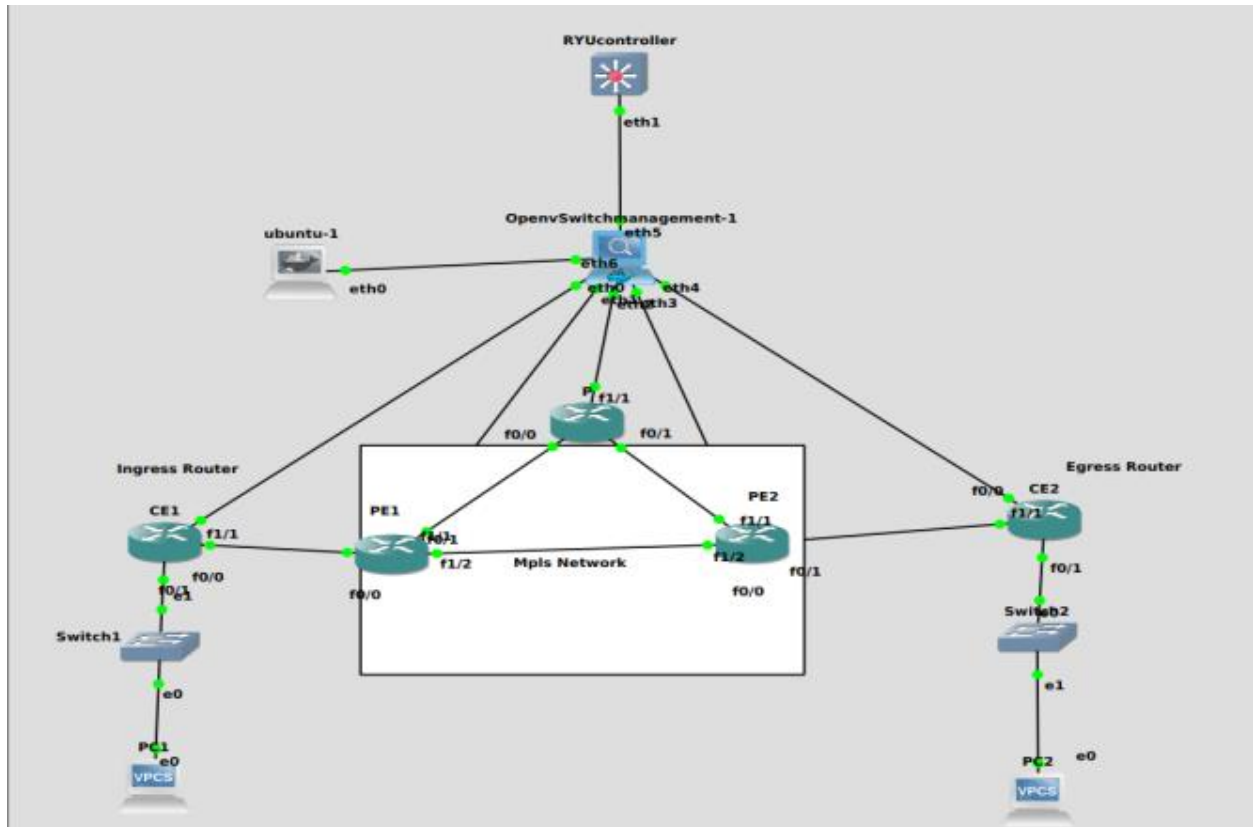


Figure 4.1: GNS3 with Mininet Topology

NAT is enabled on both MPLS and openvswitch that interconnects two different branches. NAT is used to ensure global communication across virtual devices and physical network. So, in this work established the communication between virtual machines containing SDN controllers and cisco routers in MPLS domain through NAT. CE1, CE2, PE1, PE2, and C inside GNS3 topology represent routers, and the topology shows us the physical integration of MPLS with openvswitches, whereas the topology inside mininet does not show physical integration of MPLS with openflow networks.

### 4.3 Simulation Tools and Techniques

In this thesis, make use of different software solutions, Mininet, Cbench or iperf, and Wireshark, to facilitate the development and building of the methodology.

#### 4.3.1 Installation Ryu controller

To begin with ensure your system has python installed, by executing python on the terminal, which would turn into the python shell, besides displaying the version.

➤ *Sudo pip3 install ryu*

### 4.3.2 Installation and version

- *Ryu-manager --version*

```
habteq@ryu-controller:~$ ryu-manager --version
ryu-manager 4.34
habteq@ryu-controller:~$
```

Figure 4.2: Ryu-manager version

➤ *sudo mn --version*

```
habteq@ryu-controller:~$ sudo mn --vers
2.3.0.dev6
habteq@ryu-controller:~$
```

Figure 4.3: Mininet Versions

➤ *python3 --version*

```
habteq@ryu-controller:~$ python3 --version
Python 3.8.10
habteq@ryu-controller:~$
```

Figure 4.4: Python version

➤ *ovs-vsctl --version*

```
habteq@ryu-controller:~$ ovs-vsctl --vers
ovs-vsctl (Open vSwitch) 2.13.8
DB Schema 8.2.0
habteq@ryu-controller:~$
```

Figure 4.5: Open vSwitch and Database Schema Version

The RYU controller operates as a software component responsible for the administration of network devices. Its core function revolves around the management and control of the network and its corresponding devices through the utilization of the OpenFlow protocol. The performance evaluation of the system is conducted using a testbed including a controller and a specific number of ingress router, Core MPLS, and Egress router directly linked to the controller, as depicted in Figure 4.1. The Ryu controller gets instantiated using a script called Ryu-manager. In

this work next add this to the python path variable, so that we don't need to give the complete path to invoke the controller at all times: Now to test the installation and using four terminals. In the first one, get the controller up and running:

1. In Terminal One

- Ryu-manager `ryu.app.simple_switch_13`. This starts the ryu controller with a `simple_switch` script that pushes down rules to the router. In case of your own NetApp (Ryu), replace `simple_switch.py` with your own `.py` file, with its complete path.
- In case you see an `oslo.config` error, try, as shown here:

- `sudo pip install oslo.config`

2. Wireshark run testing from h1 to h2

- `sudo wireshark &`

3. In terminal three

- `sudo mn - - controller =remote, ip=127.0.0.1 - - mac - - switch= ovsk, protocols= OpenFlows13 - - topo =linear,8`

```

habteg@ryu-controller:~$ sudo mn --controller=remote,ip=127.0.0.1--mac --switch
-ovsk,protocols=OpenFlow13 --topo=linear,8
[sudo] password for habteg:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1--mac:6653
Unable to contact the remote controller at 127.0.0.1--mac:6633
Setting remote controller to 127.0.0.1--mac:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (h5, s5) (h6, s6) (h7, s7) (h8, s8) (s2, s1
) (s3, s2) (s4, s3) (s5, s4) (s6, s5) (s7, s6) (s8, s7)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 8 switches
s1 s2 s3 s4 s5 s6 s7 s8 ...
*** Starting CLI:
mininet>

```

Figure 4.6: linear topology Installation

```

*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s22 s23 s24 s25 s26 s27 s28 s29 s30 s31 s32
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (h5, s5) (h6, s6) (h7, s7) (h8, s8) (h9, s9) (h10, s10) (h11, s11) (h12, s12) (h13, s13) (h14, s14) (h15, s15) (h16, s16) (h17, s17) (h18, s18) (h19, s19) (h20, s20) (
h21, s21) (h22, s22) (h23, s23) (h24, s24) (h25, s25) (h26, s26) (h27, s27) (h28, s28) (h29, s29) (h30, s30) (h31, s31) (h32, s32) (s2, s1) (s3, s2) (s4, s3) (s5, s4) (s6, s5) (s7, s6) (s8, s7) (s9, s8)
(s10, s9) (s11, s10) (s12, s11) (s13, s12) (s14, s13) (s15, s14) (s16, s15) (s17, s16) (s18, s17) (s19, s18) (s20, s19) (s21, s20) (s22, s21) (s23, s22) (s24, s23) (s25, s24) (s26, s25) (s27, s26) (s28,
s27) (s29, s28) (s30, s29) (s31, s30) (s32, s31)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32
*** Starting controller
c0
*** Starting 32 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s22 s23 s24 s25 s26 s27 s28 s29 s30 s31 s32 ...
*** Starting CLI:
mininet>

```

Figure 4.7: linear topology Installation 32 switches

#### 4. In terminal four

Next upgrade Open vSwitch switches so that they are good to go with OpenFlow version 1.3 scripts that we may run later.

- `sudo ovs-ofctl -O OpenFlow13 dump-flows s1`

```
Controller "tcp:127.0.0.1--mac:6653"  
Controller "ptcp:6658"  
fail_mode: secure  
Port s5-eth2  
    Interface s5-eth2  
Port s5-eth3  
    Interface s5-eth3  
Port s5  
    Interface s5  
        type: internal  
Port s5-eth1  
    Interface s5-eth1  
ovs_version: "2.13.8"
```

Figure 4.8: Open vSwitch connections

Mininet allows users to quickly create a realistic virtual network topology with hosts, router, links, and controller's. It provides a platform for rapid prototyping and testing of software-defined networks (SDN). Mininet also provides a simple interface for creating and controlling virtual networks, allowing users to test their network applications on a virtual network in seconds, while also providing the option of connecting the virtual network to a physical network. Mininet's powerful scripting capabilities allow users to easily customize their virtual networks, making it an invaluable tool for testing, debugging, and developing SDN applications. It has also supports a variety of programming languages, including Python, C, and Java, making it easy for developers to quickly develop and test their applications.

#### 4.3.2.1 Mininet Installation

- ✚ Use the command: `sudo apt-get install mininet`
- ✚ `Pingall`
- ✚ `sudo apt-get install git`
- ✚ `Git clone https://github.com/mininet/mininet`
- ✚ `Sudo apt install python3-pip`
- ✚ `Sudo pip3 install mininet`
- ✚ `Sudo apt-get install python3-tk`

```
habteg@ryu-controller:~$ sudo apt-get install mininet
[sudo] password for habteg:
Reading package lists... Done
Building dependency tree
Reading state information... Done
mininet is already the newest version (2.2.2-5ubuntu1).
The following packages were automatically installed and are no longer required:
 gir1.2-goa-1.0 libfprint-2-tod1 libfwupdplugin1 libllvm9 libxmlb1
 ubuntu-system-service
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 80 not upgraded.
```

```
✚ sudo apt-get install xterm
```

Figure 4.9: Mininet Installation

```
✚ Sudo python3 mininet/examples/miniedit.py
```

#### 4.4 Message establishment between a switch and a controller

The switch uses the controller's IP address, which is typically the Loopback interface 127.0.0.1, and the default port 6633 to establish a TCP connection with the OpenFlow controller. The foundation for creating a successful OpenFlow communication channel is this TCP connection. The controller can effectively manage the flow entries made by the switch and the behavior of the entire network. Using Wireshark or comparable network analysis software might be helpful for identifying and fixing problems. These tools make it easier to capture and examine the particular packets exchanged during this process.

#### 4.5 Messages exchanged between two hosts

In order to demonstrate host-to-host connectivity in an OpenFlow network, in this work used the Ping tool to send ICMP packets from host h1 to host h2 and the other way around. In order to find the MAC address of h2, h1 sends an ARP request to the switch, starting this procedure. Because there is no predetermined path of action for this packet, the switch encapsulates it as a PACKET-IN message and sends it to the controller.

In response, the controller creates a PACKETOUT message with an action directive inside that tells the switch to broadcast the packet to every port but the one it was received through, and then wait for a matching response. When h2 responds to this request, the switch relays this response back to the controller under the controller's direction.

The controller sends a FLOW-MOD message to the switch after receiving the ARP response from the switch. By creating a new flow entry with this message, the switch will immediately transmit any upcoming ARP replies from h2 that are meant for h1 without the need for controller action. When h1 sends an ICMP request and receives a response, a parallel process occurs. The same thing happens when h2 sends an ARP request to find h1's MAC address, followed by an ARP reply.

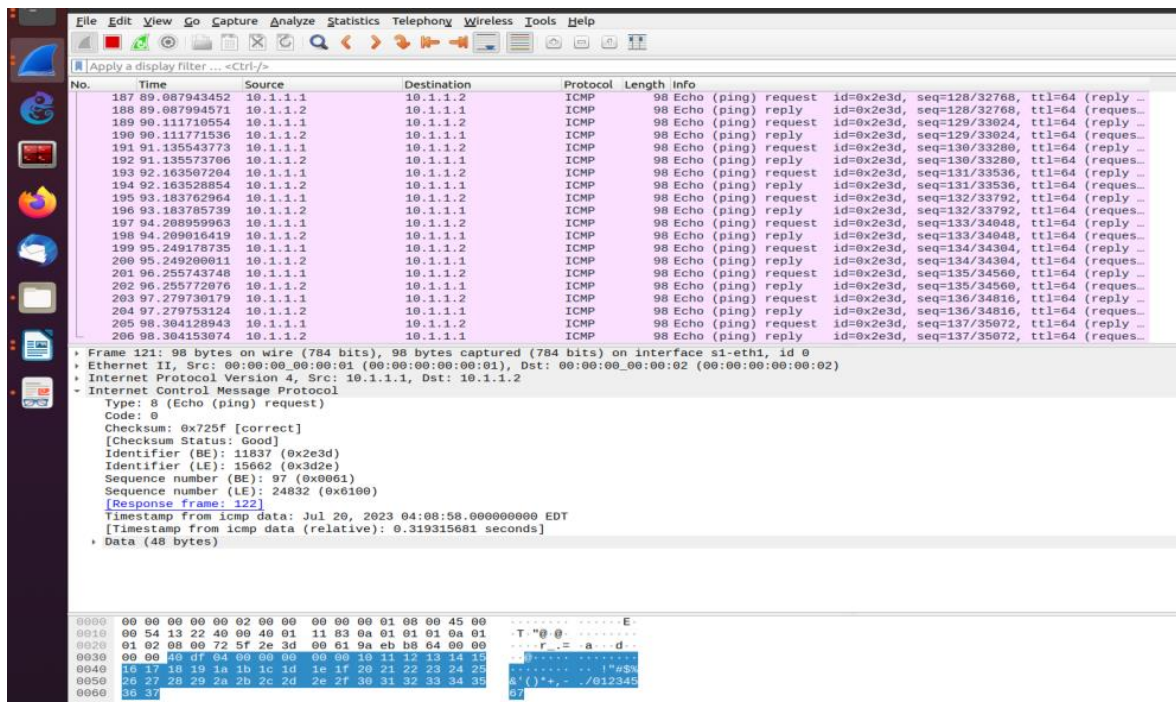


Figure 4.10: Wireshark testing from h1 to h2

## 4.6 Performance Measuring Metrics

To measure the performance of the network, in this research used only two metrics that have significant effects on the overall performance of the network. Particularly in this study, throughput, and latency are given strong focus since they are commonly used in the process of measuring the overall performance of the network.

The performance of the Ryu controller was evaluated using the Cbench tool, which measures throughput and latency by sending messages to the controller following the OpenFlow protocol and monitoring the arrival of Flow\_mod messages. The Cbench tool calculates throughput and latency using the following methods

1. The throughput is determined by counting the total number of (Flow\_mod) messages transmitted by the controller and the latency Cbench measures the time taken by the controller to process each message from the moment it is received until the corresponding Flow\_mod message is sent and or latency is determined by calculating the average time taken for message processing across multiple iterations.
2. Cbench initiates the transmission of multiple packets to the controller (Packet\_in) and awaits the controller's response by sending (Flow\_mod) messages. This process is repeated several times, and subsequently, the latency value is calculated.

Table 4.1: Cbench Running Options

Option	Description	Default Values
-c/--Controller	Controller by his name	"localhost"
-d/--debug	enable debugging	off
-l/--loops	loops per test	16
-M/--mac-addresses	unique source MAC addresses per switch	100000
-m/--ms-per test	The test time length in ms	1000
-p/--port	controller port number	6633
-s/--switches	Number of switches	
-t/--throughput	test throughput instead of latency	
-i/--connect-delay to the controller	<int>delay between groups of switches connecting	in ms
-I/--connect-group-size	<int> number of switches in a connection delay	-1-
-L/--learn-dst-macs macs before testing	send gratuitous ARP replies to learn destination (on)	-on-

#### 4.6.1 Throughput

Throughput refers to the amount of data that can be transmitted or processed within a given time period. It represents the capacity or speed at which data can flow through the network. SDN separates the control plane (which makes decisions about how the network should operate) from the data plane (which forwards network traffic).



In terms of throughput, SDN can provide several benefits. The centralized control allows for efficient resource allocation and traffic engineering, enabling better utilization of network resources and improved performance. SDN controllers can dynamically adjust network paths, prioritize traffic, and optimize routing decisions, which can enhance throughput. The throughput is commonly expressed in units such as bits per second (bps), data packets per second (pps), or data packets per time slot. A higher throughput is generally preferred in a communication system as it signifies increased data transfer capacity.

$$\text{Throughput} = \frac{\sum \text{Number of transferred bits}}{\text{Time(s)}} \text{ bps} \text{----- (Equation 1)}$$

The throughput can be defined as the ratio of the total number of packets transmitted during a specific time period. It can be calculated by determining the difference between the packet transmission time at the source node and the time of receipt at the destination node. This calculation provides an estimation of the throughput in terms of the number of packets successfully transmitted between the source and destination.

`./cbench -c localhost -p 6653 -s 1/2/4/8 -l 8 -t`

#### **4.6.1.1 Throughput Result**

The integration of MPLS with openflow improved the throughput of the network. In this thesis, enabled packet processing and monitoring on both the control plane and the data plane. In this work enabled MPLS to operate with SDN controllers it can handle low level traffic flows by the help of standalone switch, and the low level flows are directed to MPLS pipe lines instead of being sent to SDN controller's. So, this approach reduced the load of the controllers during simultaneous transmission of packets among all classes of users, and then packet transmission becomes faster and faster.

Table 4.2: Cbench Throughput result with and without MPLS network (Normal SDN)

Switch	Total throughput with MPLS Network	Total throughput without MPLS Network	Min with MPLS	Max with MPLS	Ave with MPLS	Min without MPLS	Max without MPLS	Ave without MPLS
1	7,140 Flows Per second	6,850 Flows Per second	3234.71	4073.57	3838.07	2527.03	3012.47	2759.13
2	6,730 Flows Per second	6,470 Flows Per second	3411.17	3527.70	3481.57	2921.91	3468.63	3162.78
4	4,800 Flows Per second	4,100 Flows Per second	3704.10	3904.80	3804.50	3033.02	3181.70	3107.38
8	3,560 Flows Per second	3,020 Flows Per second	4365.10	4565.00	4465.30	3765.30	3965.90	3853.60
16	2,970 Flows Per second	2,600 Flows Per second	4352.17	4751.17	4451.17	3451.17	3651.17	3531.17

```

habte@habte-VirtualBox:~$ cbench -c 127.0.0.1 -p 6653 -l 16 -s 1 -t
cbench: controller benchmarking tool
running in mode 'throughput'
connecting to controller at 127.0.0.1:6653
faking 1 switches offset 1 : 16 tests each; 1000 ms per test
with 100000 unique source MACs per switch
NOT learning destination mac addresses before the test
starting test with 0 ms delay after features_reply
ignoring first 1 "warmup" and last 0 "cooldown" loops
connection delay of 0ms per 1 switch(es)
maximum number of requests sent to controller per test is 2147483647
debugging info is off
11:02:07.987 1 switches: response/requests: 3060/23967 total = 3.059559 per ms
11:02:09.090 1 switches: response/requests: 3581/3196 total = 3.568923 per ms
11:02:10.195 1 switches: response/requests: 3936/4794 total = 3.917915 per ms
11:02:11.296 1 switches: response/requests: 4001/3995 total = 3.999324 per ms
11:02:12.399 1 switches: response/requests: 3243/3196 total = 3.234706 per ms
11:02:13.501 1 switches: response/requests: 3300/3196 total = 3.292418 per ms
11:02:14.603 1 switches: response/requests: 4081/3995 total = 4.073570 per ms
11:02:15.703 1 switches: response/requests: 3943/3995 total = 3.942929 per ms
11:02:16.804 1 switches: response/requests: 3976/3196 total = 3.972659 per ms
11:02:17.906 1 switches: response/requests: 4002/3995 total = 3.994271 per ms
11:02:19.006 1 switches: response/requests: 4001/3995 total = 4.000652 per ms
11:02:20.108 1 switches: response/requests: 3936/4794 total = 3.928822 per ms
11:02:21.211 1 switches: response/requests: 4000/3995 total = 3.989504 per ms
11:02:22.311 1 switches: response/requests: 3808/3995 total = 3.807547 per ms
11:02:23.412 1 switches: response/requests: 3949/3995 total = 3.948021 per ms
11:02:24.517 1 switches: response/requests: 3921/3196 total = 3.899839 per ms
Total Average: responses/requests = 6850/11900
RESULT: 1 switches 15 tests min/max/avg/stddev = 2527.03/3012.47/2759.13/251.66 responses/s
habte@habte-VirtualBox:~$

```

Figure 4.11: Cbench Throughput Result without MPLS one switch

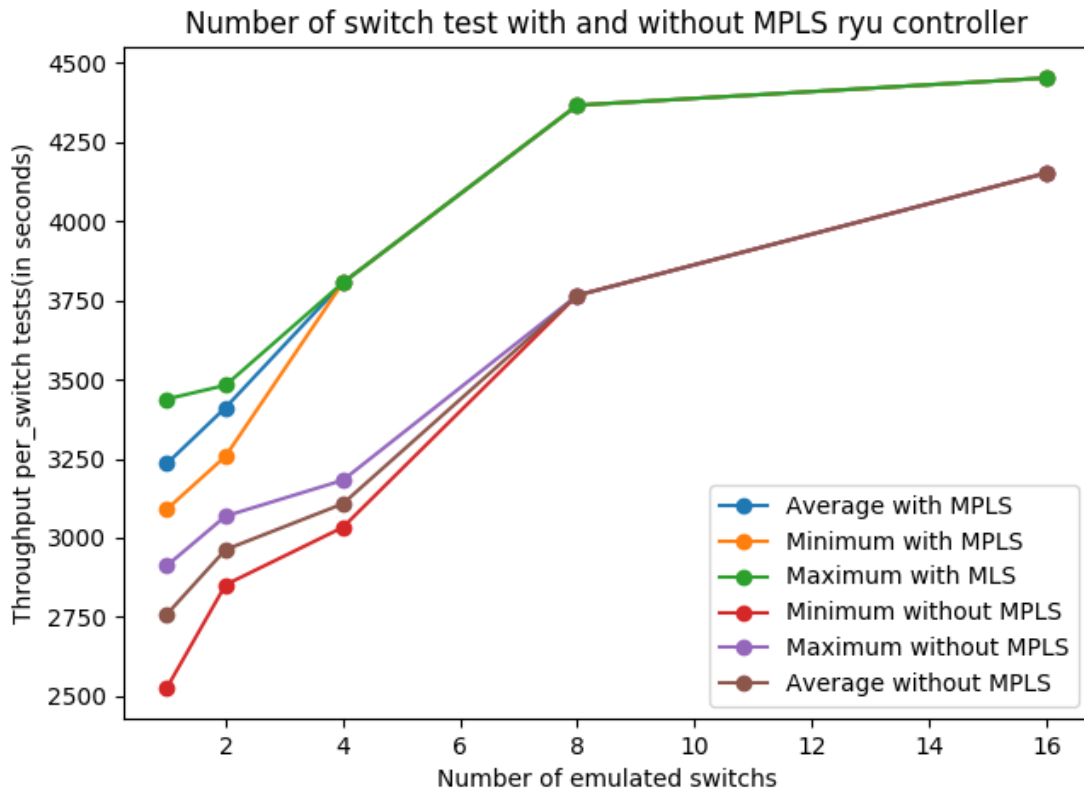


Figure 4.12: Number of switch Tests with and without MPLS Ryu controller

```

DBU_20.04 [Running] - Oracle VM VirtualBox
Activities Terminal
habte@habte-VirtualBox:~$ cbench -c 127.0.0.1 -p 6653 -MPLS -l 16 -m 2000 -s 1 -t
cbench: controller benchmarking tool
running in mode 'throughput'
connecting to controller at 127.0.0.1:6653
faking 1 switches offset 1 :: 16 tests each; 20000 ms per test
with 100 unique source MACs per switch
NOT learning destination mac addresses before the test
starting test with 0 ms delay after features_reply
ignoring first 1 "warmup" and last 0 "cooldown" loops
connection delay of 0ms per 1 switch(es)
maximum number of requests sent to controller per test is 2147483647
debugging info is off
07:58:29.907 1 switches: response/requests: 64065/85490 total = 3.201368 per ms
07:58:50.009 1 switches: response/requests: 60256/59925 total = 3.012466 per ms
07:59:10.118 1 switches: response/requests: 54625/54332 total = 2.730000 per ms
07:59:30.219 1 switches: response/requests: 54108/54332 total = 2.705399 per ms
07:59:50.322 1 switches: response/requests: 59795/59925 total = 2.989312 per ms
08:00:10.425 1 switches: response/requests: 58192/58327 total = 2.909281 per ms
08:00:30.525 1 switches: response/requests: 56400/56729 total = 2.819948 per ms
08:00:50.626 1 switches: response/requests: 58768/58327 total = 2.938377 per ms
08:01:10.733 1 switches: response/requests: 53648/53533 total = 2.681465 per ms
08:01:30.838 1 switches: response/requests: 54176/54332 total = 2.708139 per ms
08:01:50.947 1 switches: response/requests: 54224/54332 total = 2.709921 per ms
08:02:11.051 1 switches: response/requests: 55087/54332 total = 2.753833 per ms
08:02:31.155 1 switches: response/requests: 51298/51935 total = 2.564448 per ms
08:02:51.263 1 switches: response/requests: 50559/50337 total = 2.527031 per ms
08:03:11.366 1 switches: response/requests: 52216/51935 total = 2.610489 per ms
08:03:31.466 1 switches: response/requests: 54536/54332 total = 2.726789 per ms
Total Average: responses/requests = 7140.00/57028.44
RESULT: 1 switches 15 tests min/max/avg/stdev = 3204.71/4073.57/3838.07/142.66 responses/s
habte@habte-VirtualBox:~$

```

Figure 4.13: Cbench Throughput Result with MPLS one switch

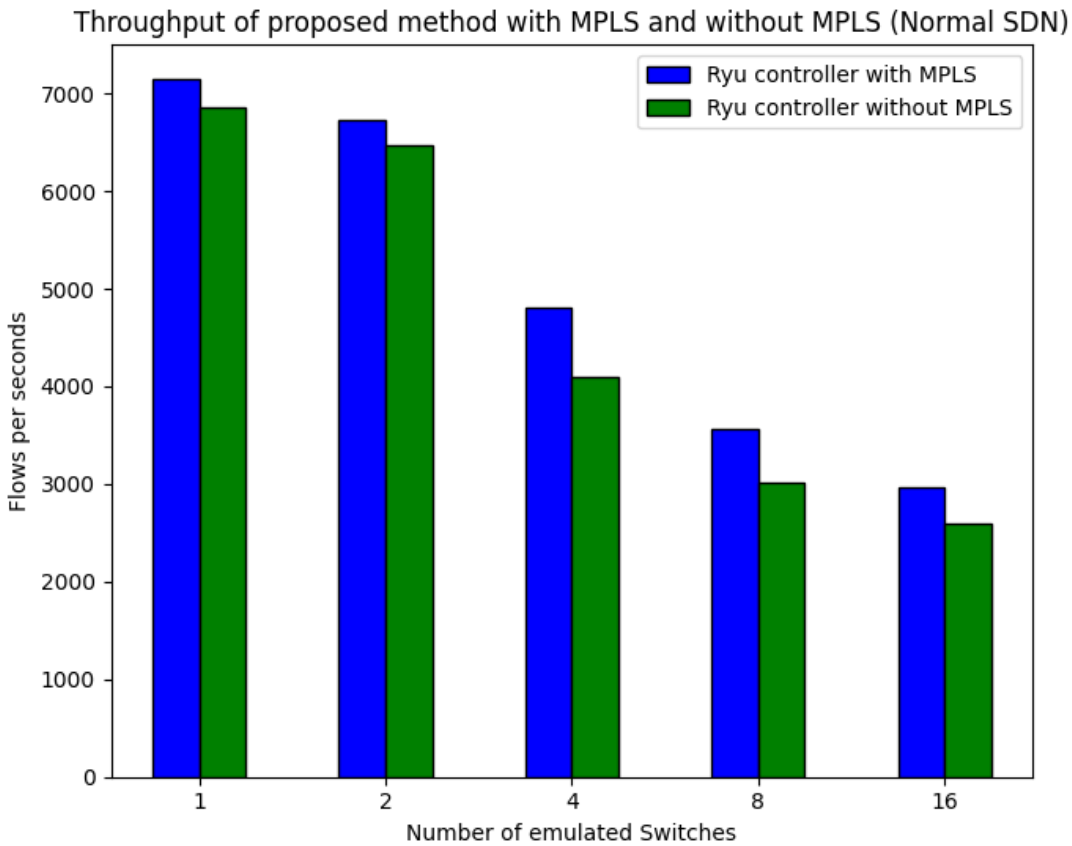


Figure 4.14: Throughput of proposed method with MPLS and normal SDN

### Discussion

The above figure 5.10 illustrates if the number of switches are devices that allow network packets to be forwarded between different network segments. In networking, flows per second refers to the time it takes for a request to be processed and a response to be sent back. Both the average throughput of With MPLS and without kept slightly decrease based on number of switch, as shown in the above graph. The achieved maximum throughput of the proposed method clearly showcases its exceptional performance, demonstrating that when connecting to a single switch results in maximum 7,148 flows per second. And the Ryu controller without MPLS network has the maximum throughput for a single switch 6,850 flows per second. The proposed method, which includes MPLS (Multiprotocol Label Switching), demonstrates superior performance compared to the normal SDN implementation, particularly in terms of throughput.

When evaluating network performance, throughput is a critical metric that measures the amount of data transmitted successfully over the network within a given timeframe. MPLS, being a high-

performance protocol designed to efficiently route and forward data packets using labels, significantly enhances the network's ability to handle data traffic and reduces the processing overhead on switches and routers.

#### 4.6.2 Latency

Latency refers to the amount of time it takes for the controller to process a network event or execute an action in response to a request or event received from the switches in the network. SDN controller can be measured by the time delay between the occurrence of a network event, such as a packet arrival or a switch status change, and the controller's response to that event. It includes the time taken for message parsing, flow table lookup, decision-making, and sending instructions back to the switches. The general formula to calculate response time is expressed as follow

```
./cbench -c localhost -p 6653 -s 1/2/4/8 -l 4
```

##### 4.6.2.1 Latency Result

The ability of handling packets on both the data plane and the control plane allowed network operations to be quicker and quicker because the controllers are no more be overloaded, and low level packets are handled by the switches by the help of MPLS. MPLS shares the load of the controller when there is simultaneous packet transmission among the groups of users; not only this, having one controller on the control plane and the controller are no longer be in a busy tone, as a result, there is no more queuing delay.

Table 4.3 Latency total result with and without MPLS Network

Switch	Total latency with MPLS Network	Total latency without MPLS Network	Min with MPLS	Max with MPLS	Ave with MPLS	Min WO MPLS	Max WO MPLS	Ave WO MPLS
1	1.123 per ms	1.189 per ms	0.853	1.107	0.977	1.000	1.290	1.153
2	1.300 per ms	1.325 per ms	0.843	0.843	0.843	1.281	1.325	1.300
4	1.320 per ms	1.368 per ms	0.811	0.811	0.811	1.356	1.452	1.411
8	1.320 per ms	1.534 per ms	0.811	0.811	0.811	1.459	1.562	1.519
16	1.320 per ms	1.642 per ms	0.811	0.811	0.811	1.538	1.701	1.608

```

DBU_20.04 [Running] - Oracle VM VirtualBox
Activities Terminal habte
habte@habte-VirtualBox:~$ cbench -c 127.0.0.1 -p 6653 -s 1 -l 16
cbench: controller benchmarking tool
running in mode 'latency'
connecting to controller at 127.0.0.1:6653
faking 1 switches offset 1 :: 16 tests each; 1000 ms per test
with 100000 unique source MACs per switch
NOT learning destination mac addresses before the test
starting test with 0 ms delay after features_reply
ignoring first 1 "warmup" and last 0 "cooldown" loops
connection delay of 0ms per 1 switch(es)
maximum number of requests sent to controller per test is 2147483647
debugging info is off
07:52:29.077 1 switches: response/requests: 992/993 total = 0.991887 per ms
07:52:30.181 1 switches: response/requests: 1211/1211 total = 1.210202 per ms
07:52:31.323 1 switches: response/requests: 1206/1206 total = 1.205999 per ms
07:52:32.423 1 switches: response/requests: 1276/1276 total = 1.275562 per ms
07:52:33.524 1 switches: response/requests: 1104/1104 total = 1.103751 per ms
07:52:34.688 1 switches: response/requests: 1205/1205 total = 1.204643 per ms
07:52:35.789 1 switches: response/requests: 600/600 total = 0.599999 per ms
07:52:36.896 1 switches: response/requests: 789/789 total = 0.787193 per ms
07:52:37.997 1 switches: response/requests: 1273/1273 total = 1.272999 per ms
07:52:39.097 1 switches: response/requests: 1264/1264 total = 1.263999 per ms
07:52:40.199 1 switches: response/requests: 1186/1186 total = 1.184278 per ms
07:52:41.299 1 switches: response/requests: 1276/1276 total = 1.275999 per ms
07:52:42.407 1 switches: response/requests: 1233/1233 total = 1.232999 per ms
07:52:43.526 1 switches: response/requests: 1292/1292 total = 1.290223 per ms
07:52:44.670 1 switches: response/requests: 1211/1211 total = 1.210999 per ms
07:52:45.786 1 switches: response/requests: 1183/1183 total = 1.182999 per ms
Total Average: responses/requests = 1.189/2000
RESULT: 1 switches 15 tests min/max/avg/stddev = 1.000/1.290/1.153/189.41 responses/ms
habte@habte-VirtualBox:~$

```

Figure 4.15: Cbench Latency Result without MPLS one switch

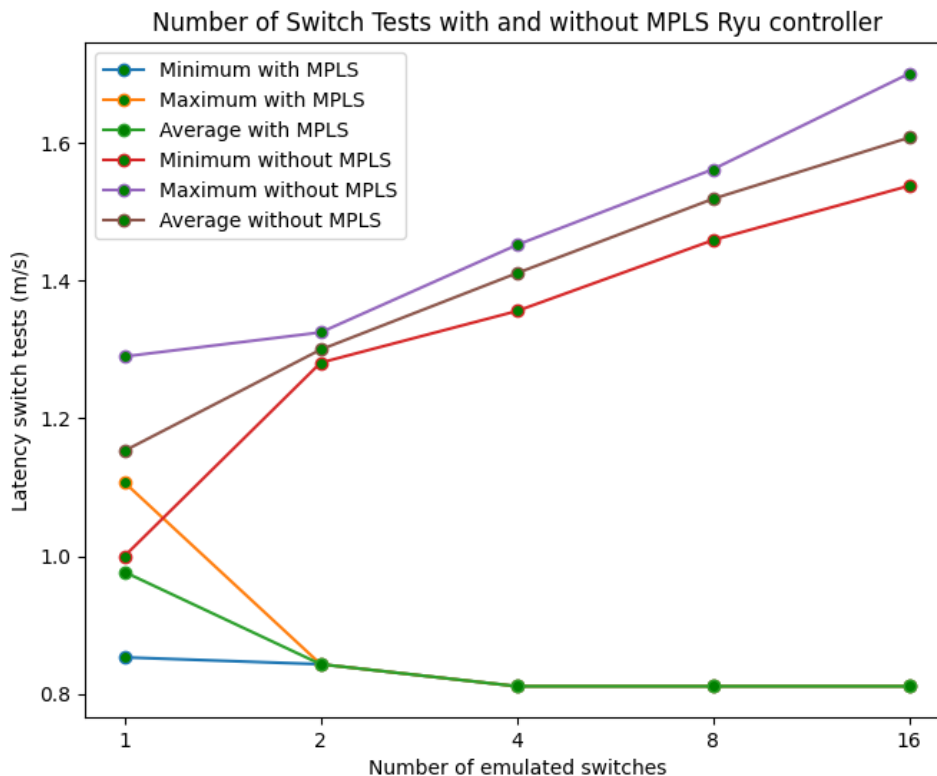


Figure 4.16: Cbench Latency switch tests with and without MPLS result

```

habte@habte-VirtualBox:~$ cbench -c 127.0.0.1 -p 6653 -MPLS -l 16 -m 20000 -s 1
cbench: controller benchmarking tool
running in mode 'latency'
connecting to controller at 127.0.0.1:6653
faking 1 switches offset 1 :: 16 tests each; 20000 ms per test
with 100 unique source MACs per switch
NOT learning destination mac addresses before the test
starting test with 0 ms delay after features_reply
ignoring first 1 "warmup" and last 0 "cooldown" loops
connection delay of 0ms per 1 switch(es)
maximum number of requests sent to controller per test is 2147483647
debugging info is off
07:44:25.344 1 switches: response/requests: 23939/23940 total = 1.196950 per ms
07:44:45.444 1 switches: response/requests: 21527/21527 total = 1.076350 per ms
07:45:05.555 1 switches: response/requests: 17754/17754 total = 0.887700 per ms
07:45:25.656 1 switches: response/requests: 18204/18204 total = 0.910196 per ms
07:45:45.776 1 switches: response/requests: 17291/17291 total = 0.864550 per ms
07:46:05.878 1 switches: response/requests: 17814/17814 total = 0.890631 per ms
07:46:25.979 1 switches: response/requests: 21733/21733 total = 1.086649 per ms
07:46:46.080 1 switches: response/requests: 19314/19314 total = 0.965671 per ms
07:47:06.198 1 switches: response/requests: 17065/17065 total = 0.853244 per ms
07:47:26.371 1 switches: response/requests: 19329/19329 total = 0.966450 per ms
07:47:46.476 1 switches: response/requests: 19243/19243 total = 0.962150 per ms
07:48:06.579 1 switches: response/requests: 18295/18295 total = 0.914604 per ms
07:48:26.707 1 switches: response/requests: 20810/20810 total = 1.040482 per ms
07:48:46.807 1 switches: response/requests: 20877/20877 total = 1.043850 per ms
07:49:06.908 1 switches: response/requests: 21936/21936 total = 1.096800 per ms
07:49:27.010 1 switches: response/requests: 22148/22148 total = 1.107347 per ms
Total Average: responses/requests = 1.123/2000
RESULT: 1 switches 15 tests min/max/avg/stdev = 0.853/1.107/0.977/87.15 responses/ms
habte@habte-VirtualBox:~$

```

Figure 4.17: Latency of Ryu controller with MPLS

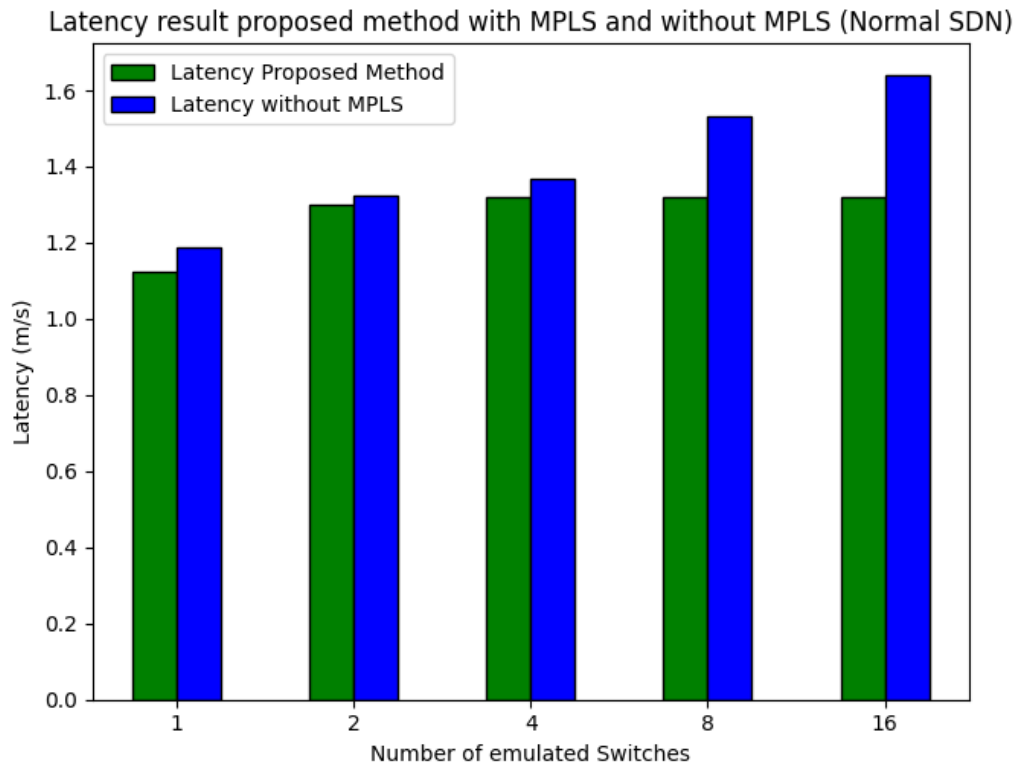


Figure 4.18: Latency Result proposed method and Normal SDN

In this section, analyze the test results to compare the performance enhancements of the Ryu controller with MPLS integration and normal SDN. As shown in the above figure, the Ryu controller with MPLS demonstrated a latency of 1.123 ms, whereas the Ryu controller without MPLS exhibited a latency of 1.189 ms, the latency results indicate that the Ryu controller with MPLS had the highest latency among the tested result. According to [38] the experiment setup cbench was used to emulate different number of switches (1, 2, 4, 8, 16, and 32) which connect to the Ryu controller. The objective was to evaluate the network throughput and latency performance evaluation. The result shows that the above figure that performance is increase independent of the number of switches emulated. The researcher only one switch is greater latency than proposed method but the rest of the switch has a better latency performance using the Ryu controller with MPLS integration.

### 4.6.3 Jitter

Jitter refers to the variation in the delay of received packets in a network [33], it is a measure of the variability in the time it takes for data packets to travel from the sender to the receiver. Measured in milliseconds (ms) or microseconds ( $\mu$ s), representing the time variation in packet arrival.

- Install iperf : *Sudo apt-get install iperf*
- Start the Ryu-Controller: *ryu-manager ryu.app.simple\_switch\_13*
- Create the topology : *sudo mn - - controller =remote, ip=127.0.0.1 - - mac - - switch= ovsk, protocols= OpenFlows13 - - topo =single,4*
- **Verify iperf Server:** Inside the Mininet CLI, you should see the iperf server running on the specified host  
*H2: iperf -s -u -p 6653 -l 1* - This command starts the iperf server
- **Test with iperf Client:** On another host in the Mininet environment, start an iperf client to test:  
*H1: iperf -c <iperf\_server\_ip> -u -b 10M /20M/30M/40M/50M -t 15 -p 6653*



Table 4.4: Jitter Running Options

Option	Description
-s	Acts as a server
-c	Acts as a client
-u	UDP test
-p	Port number of the controller
-b	Bandwidth test <i>10M/20M/30M/40M/50M</i>
-i	Iteration of testing results
6653	Port numbers of the RYU controller

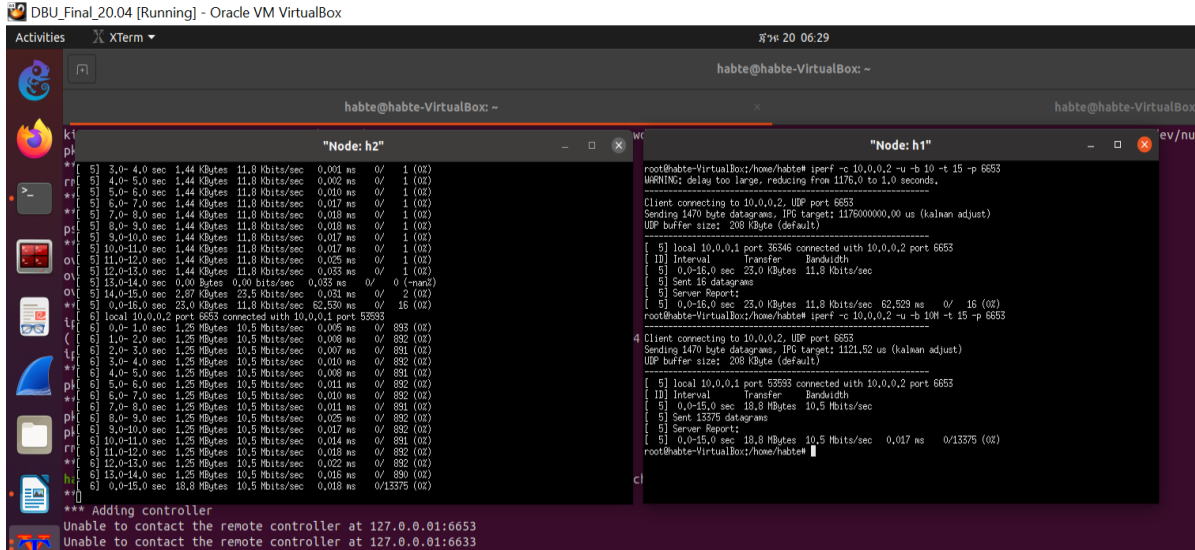


Figure 4.19 : Jitter result h1 to h2

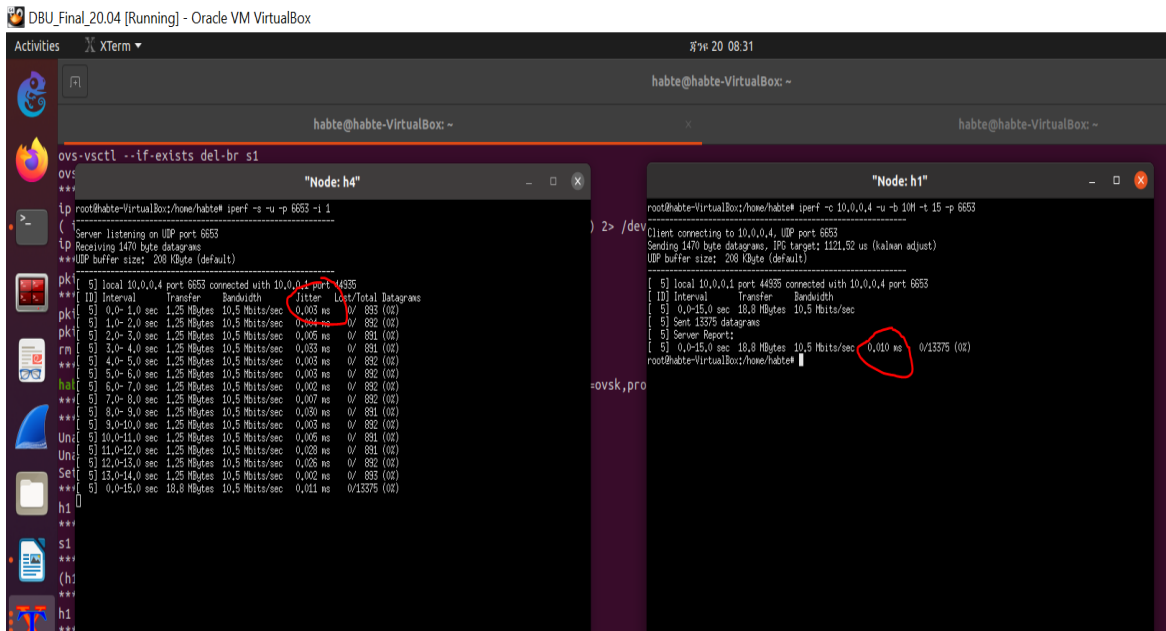


Figure 4.20: Jitter result h1 to h4

Table 4.5 Result Jitter Single Topology

Bandwidth (Mbits/sec)	<i>h</i> 1 to <i>h</i> 2 jitter (ms)	<i>h</i> 1 to <i>h</i> 3 jitter (ms)	<i>h</i> 1 to <i>h</i> 4 jitter (ms)	<i>h</i> 2 to <i>h</i> 3 jitter (ms)	<i>h</i> 2 to <i>h</i> 4 jitter (ms)	<i>h</i> 3 to <i>h</i> 4 jitter (ms)
10	0.017ms	0.013 ms	0.010 ms	0.016 ms	0.017 ms	0.013 ms
20	0.012ms	0.010 ms	0.009 ms	0.010ms	0.009 ms	0.012 ms
30	0.007ms	0.005 ms	0.007 ms	0.007ms	0.007 ms	0.005 ms
40	0.011ms	0.005 ms	0.004 ms	0.011ms	0.005 ms	0.005 ms
50	0.011ms	0.011 ms	0.009 ms	0.011ms	0.011 ms	0.008 ms

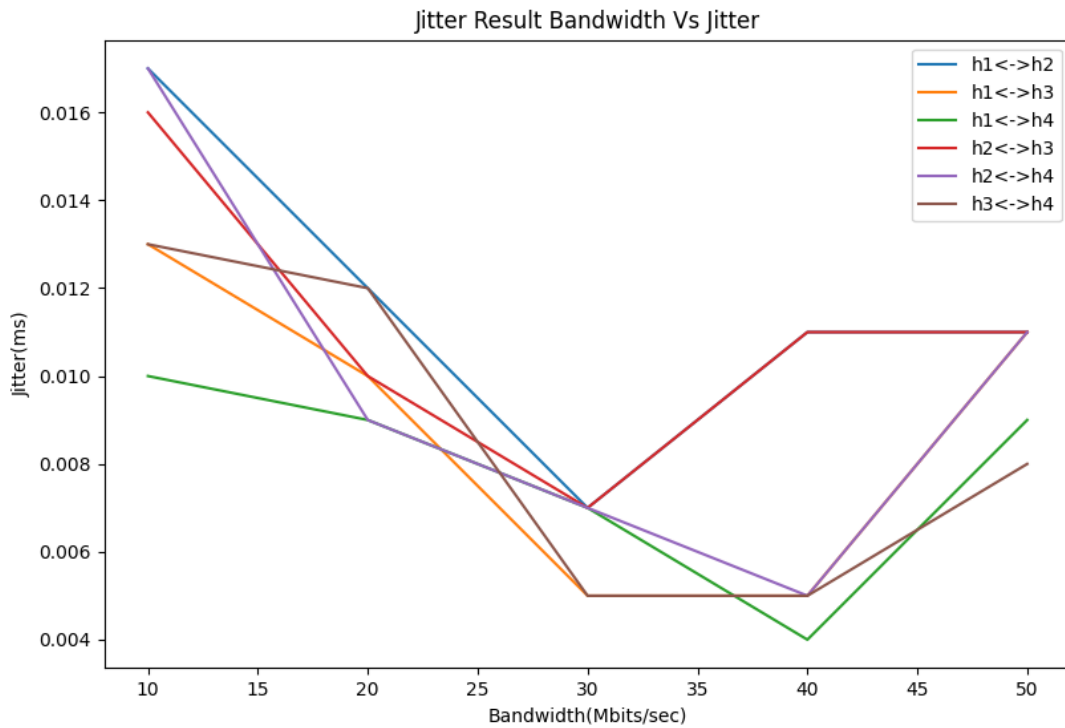


Figure 4.21: Jitter Result Bandwidth vs Jitter

Table 4.6: Shows the overall performance metrics used in this research study as compared to previous works

Authors	Protocol used	Topology	Nature of SDN	Simulation Tools	Evaluation Tools	Performance Evaluation Metrics	SDN Controllers used
Alaa Taima Abu-Salih [36]	Openflow	Single, linear	Pure SDN	Mininet	Cbench	Throughput and latency	OpenFlow, Ryu controller
Lusani Mamushiane, Albert Lysko, Sabelo Dlamini [23]	Openflow	Single, linear plus MAC testing	Pure SDN	Mininet	Cbench	Throughput and latency	OpenFlow, Ryu, floodlight, Opendaylight, & ONOScontroller
Shavan Askar & Faris Ketil.[6]	Openflow	Single	Pure SDN	Mininet	iperf3 benchmark	Throughput & RTT using TCP,UDP,ICMP traffic	OpenFlow, Pox and Ryu Controller
Bhardwaj, S., & Panda, S. N. [9]	Openflow	single	Pure SDN	Mininet	Cbench	Throughput, Band width, Round trip time (RTT) and Transmission Open flow Packets	OpenFlow, Ryu Controller
Islam, M. T., Islam, N., & Refat, M. Al.[18]	Openflow	Single	Pure SDN	Mininet	Cbench	Throughput, Round-Trip Time( RTT), Jitter and Packet lose	Open Flow, Ryu controller
Proposed Study	MPLS and Open flow	Single, linear[ 2,4,16, 32]	Pure SDN	GNS3 and mininet	Cbench	Throughput, Jitter and latency	Open Flow, Ryu controller

In the first chapter of the research, a fundamental question has raised concerning the impact of network size on data plane requests and its consequent effect on network performance as given below questions

- How to investigate the effect of with and without MPLS network on throughput, and latency using Ryu controller?
- How to compare the performance of Ryu controller without MPLS or normal SDN and with MPLS network

The implementation of an MPLS network has a significant impact on both throughput and latency. One of the most important advantages of MPLS is when the workload increases, the

network dynamically adjusts itself to handle the rising demand efficiently. This adaptability is essential in maintaining a smooth flow of data and minimizing delays in processing. Furthermore, MPLS helps to optimize the workload distribution across the network by updating the workload based on the capacity of the queue size in the OpenFlow switches. When many queues become congested, MPLS adjusts the size of data batches, which allows for more effective management of network resources and improves potential bottlenecks.

The latency of an MPLS network based on the RYU controller is especially reduced due to its ability to minimize the wait time for packets to access network resources. MPLS efficiently directs data packets along predefined paths, reducing the time taken for packets to reach their intended destinations. By prioritizing packet forwarding, MPLS effectively minimizes latency and enhances the overall responsiveness of the network.

In terms of throughput, the implementation of MPLS in a RYU controller based network leads to significant improvements. As the MPLS network efficiently handles data plane requests, there is a clear increase in the system's ability to process multiple requests simultaneously. This streamlined processing allows the RYU controller to handle job requests quickly and efficiently, without the saturation of many queues. As a result, the throughput of the MPLS-based network is improved, providing a more continuous and faster data transmission experience.

## Chapter Five

### Conclusions and Future Works

#### 5.1 Conclusions

The utilization of MPLS networks for enhancing the performance of the Ryu controller offers significant improvements in terms of throughput and latency. By using MPLS, the Ryu controller can optimize network paths, allocate resources efficiently, and ensure reliable

MPLS implements on the data plane enables techniques using Ryu controller, such as traffic shaping and load balancing. These techniques allow for increased throughput and decrease the latency. By the label routing mechanism of MPLS enables the Ryu controller to make faster routing decisions compared to without MPLS Ryu controller. Additionally, it has supports explicit paths and fast rerouting, which further contribute to minimizing latency by quickly redirecting traffic along alternative paths in case of link failures, and provides Quality of Service (QoS) mechanisms that enable the Ryu controller to prioritize traffic based on different service requirements. With MPLS, the Ryu controller can assign different labels to packets, allowing for differentiated handling of traffic flows.

The controller is located between the infrastructure layer (Data plane), which is made up of various network devices, and the application layer (Application plane). As a result, the controller is in responsibility for controlling the network's resources using the OpenFlow protocol. Based on the results illustrated in chapter four figure 4.12 or table 4.2 it can be concluded that the Ryu controller integrated with the MPLS network achieved the highest throughput, reaching in one switch 7,140 flows per second. In comparison, the throughput for the SDN controller varied only based on the number of flows set up and reached in one switch 6,850 flows per second, when connected without MPLS. Therefore, using the Ryu controller with MPLS resulted in an average throughput enhancement of 4%.

Additionally, latency tests were performed on the controllers with different routers of connected switches to observe the impact of increasing workload on switches, as shown the above figure 4.17 or table 4.3 in chapter four. The tests shown that the Ryu controller with MPLS had a latency of 1.123 ms, while the Ryu controller without MPLS had a latency of 1.189 ms. In

general, the Ryu controller with MPLS exhibited an average latency improvement of 5% compared to the Ryu controller without MPLS.

Consequently, the implementation of the Ryu controller with MPLS led to an average enhancement of 4% in throughput and 5% reduction in latency. These findings indicate that the proposed mechanism successfully improved throughput and reduced latency when compared to without MPLS integration or normal SDN.

## **5.2 Future Works**

This research successfully integrated and executed an MPLS networks to increase the performance of the Ryu controller. Moreover, with this methodology significantly optimized the controller's functionality. Futures works will aim at considering:

- We conducted our experiments on virtual machines, which limited the available resources for running applications, impacting our experimental outcomes. Therefore, we suggest transitioning to real machines.
- We have evaluated and analysis the performance metrics using latency and throughput, we suggest evaluated and analysis the performance of the RYU controller using resource utilization, fault tolerance, and bandwidth usage etc.

## References

- [1] Ominike Akpovi, A., A. O. Adebayo, and F. Y. Osisanwo. "Introduction to Software Defined Networks (SDN).."  
*International Journal of Applied Information Systems* (2016): 10-14 .<https://doi.org/10.5120/ijais2016451623>
- [2] Al Bowarab, M. H., Zakaria, N. A., & Zainal Abidin, Z. (2019). Load balancing algorithms in software defined network. *International Journal of Recent Technology and Engineering*, 7(6), 686–693.
- [3] Ali, S., Alvi, M. K., Faizullah, S., Khan, M. A., Alshanqiti, A., & Khan, I. (2020). Detecting DDoS attack on SDN Due to vulnerabilities in OpenFlow. 2019 International Conference on Advances in the Emerging Computing Technologies, AECT 2019. <https://doi.org/10.1109/AECT47998.2020.9194211>
- [4] Alraawi, Abdulmaged Ali M., and Sami Abbas Nagar Adam. "Performance evaluation of controller based sdn network over non-controller based network in data center network." In 2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE), pp. 1-4. IEEE, 2021
- [5] A sadollahi, Saleh, Bhargavi Goswami, Ahmad Sohaib Raoufy, and Hedmilson Guimaraes Jose Domingos. "Scalability of software defined network on floodlight controller using OFNet." In 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), pp. 1-5. IEEE, 2017.
- [6] Askar, S., & Ketu, F. (2021). Performance Evaluation of Different SDN Controllers: A Review. 67–80. <https://doi.org/10.5281/zenodo.4742771>
- [7] Belayeneh, T. (2021). Improving the performance of software defined Networks in multi metrics perspective , MSc Thesis. <https://doi.org/10.1201/9780203970843-50>
- [8] Belgaum, M. R., Musa, S., Alam, M. M., & Su'Ud, M. M. (2020). A Systematic Review of Load Balancing Techniques in Software-Defined Networking. *IEEE Access*, 8, 98612–98636. <https://doi.org/10.1109/ACCESS.2020.2995849>
- [9] Bhardwaj, S., & Panda, S. N. (2022). Performance Evaluation Using RYU SDN Controller in Software-Defined Networking Environment. *Wireless Personal Communications*, 122(1), 701–723. <https://doi.org/10.1007/s11277-021-08920-3>

- [10] Braun, W., & Menth, M. (2014). Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices. *Future Internet*, 6(2), 302–336. <https://doi.org/10.3390/fi6020302>
- [11] Capdevila-Werning, R. (2018). Open Networking Foundation. *SDN Architecture*, 1, 373–380. <https://doi.org/10.1002/9781119154242.ch42>
- [12] Chuang, P. J., & Chen, H. J. (2018). Efficient load balancing in software defined networks. *Proceedings of the 2017 IEEE International Conference on Information, Communication and Engineering: Information and Innovation for Modern Technology, ICICE 2017*, 12, 526–528. <https://doi.org/10.1109/ICICE.2017.8478924>
- [13] Stallings, W. (2015). *Foundations of modern networking: SDN, NFV, QoE, IoT, and Cloud*. Addison-Wesley Professional.
- [14] Liyanage, M., Gurtov, A., & Ylianttila, M. (Eds.). (2015). *Software defined mobile networks (SDMN): beyond LTE network architecture*. John Wiley & Sons.
- [15] Erel, M., Teoman, E., Özçevik, Y., Seçinti, G., & Canberk, B. (2016). Scalability analysis and flow admission control in mininet-based SDN environment. *2015 IEEE Conference on Network Function Virtualization and Software Defined Network, NFV-SDN 2015*, 18–19. <https://doi.org/10.1109/NFV-SDN.2015.7387396>
- [16] Erickson, D. (2013). The Beacon OpenFlow controller. *HotSDN 2013 - Proceedings of the 2013 ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 13–18. <https://doi.org/10.1145/2491185.2491189>
- [17] Fernandez, C., & Muñoz, J. L. (2015). *Software Defined Networking (SDN) with OpenFlow 1.3, Open vSwitch and Ryu*. 11–181. <https://upcommons.upc.edu/bitstream/handle/2117/77684/sdn-book.pdf.zip>
- [18] Islam, M. T., Islam, N., & Refat, M. Al. (2020). Node to Node Performance Evaluation through RYU SDN Controller. *Wireless Personal Communications*, 112(1), 555–570. <https://doi.org/10.1007/s11277-020-07060-4>
- [19] Lara, A., Kolasani, A., & Ramamurthy, B. (2013). *White-Energy-Cultural-Evolution.pdf*. 1–20.
- [20] Latif, Z., Sharif, K., Li, F., Karim, M. M., Biswas, S., & Wang, Y. (2020). A comprehensive survey of interface protocols for software defined networks. *Journal of Network and*



- Computer Applications, 156, 102563. <https://doi.org/10.1016/j.jnca.2020.102563>
- [21] Li, X., Mhamdi, L., & Hamdi, M. (2007). High-performance Packet Switching architectures. In *High-performance Packet Switching Architectures* (Issue January 2016). <https://doi.org/10.1007/1-84628-274-8>
- [22] Li, Y., Guo, X., Pang, X., Peng, B., Li, X., & Zhang, P. (2020). Performance Analysis of Floodlight and Ryu SDN Controllers under Mininet Simulator. 2020 IEEE/CIC International Conference on Communications in China, ICCIC Workshops 2020, 85–90. <https://doi.org/10.1109/ICCCWorkshops49972.2020.9209935>
- [23] Mamushiane, L., Lysko, A., & Dlamini, S. (2018). A comparative evaluation of the performance of popular SDN controllers. *IFIP Wireless Days, 2018-April*, 54–59. <https://doi.org/10.1109/WD.2018.8361694>
- [24] Mitiku, K. (2022). Improving the Performance of Software-Defined Network Load Balancer Using Open Flow Based Multi-Controller Topology.
- [25] Mittal, S. (2018). Performance Evaluation of Openflow SDN Controllers. In *Advances in Intelligent Systems and Computing* (Vol. 736). Springer International Publishing. [https://doi.org/10.1007/978-3-319-76348-4\\_87](https://doi.org/10.1007/978-3-319-76348-4_87)
- [26] Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys and Tutorials*, 16(3), 1617–1634. <https://doi.org/10.1109/SURV.2014.012214.00180>
- [27] Petros, B. (2019). Performance Enhancement of Floodlight Software Defined Networking Controller using Workload Adaptive Packet Batching. MSc Thesis.
- [28] Shah, S. A., Faiz, J., Farooq, M., Shafi, A., & Mehdi, S. A. (2013). An architectural evaluation of SDN controllers. *IEEE International Conference on Communications*, 1, 3504–3508. <https://doi.org/10.1109/ICC.2013.6655093>
- [29] Talavera, M. (2014). Load balancing control of a server network cluster. July.
- [30] Xue, H., Kim, K. T., & Youn, H. Y. (2019). Dynamic load balancing of software-defined networking based on genetic-ant colony optimization. *Sensors (Switzerland)*, 19(2). <https://doi.org/10.3390/s19020311>

- [31] Zakia, U., & Ben Yedder, H. (2017). Dynamic load balancing in SDN-based data center networks. 2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON 2017, 10(03), 242–247.
- [32] Zerifi, M., Ezzouhairi, A., & Boulaalam, A. (2020). Overview on SDN and NFV based architectures for IoT environments: Challenges and solutions. 4th International Conference on Intelligent Computing in Data Sciences, ICDS 2020.  
<https://doi.org/10.1109/ICDS50568.2020.9268779>
- [33] Zhu, Liehuang & Karim, Md Monjurul & Sharif, Kashif & Xu, Chang & Li, Fan & Du, Xiaojiang & Guizani, Mohsen. (2020). SDN Controllers: A Comprehensive Analysis and Performance Evaluation Study. *ACM Computing Surveys*. 53. 1-40. 10.1145/3421764.
- [34] B. Heller. Cbench [Online]. Available:<https://github.com/mininet/oflops/tree/master/cbench>. [Accessed: Jan. 5, 2019].
- [35] Alemayehu, K. (2019, December). Analyzing Impact of Segment Routing MPLS on QoS Addis Ababa University, Addis Ababa, Ethiopia.
- [36] Mohammed, T. M., Behzad , A., Nader , M., & Luca , C. (2018). SDN-Based Resource Allocation in MPLS Networks: *concurrency and computation practice and experience* (pp. 1-11). 2018 John Wiley & Sons, Ltd.
- [37] Bellessa, J. (2015). *Implementing MPLS with Label Switching in Software-Defined Networks* University of Illinois, Urbana, Illinois, USA.
- [38] Albu-Salih, A. T. (2022). Performance evaluation of ryu controller in software defined networks. *Journal of al-qadisiyah for computer science and mathematics*, 14(1), Page-1.
- [39] Zhang, Y., Cui, L., Wang, W., & Zhang, Y. (2017). A survey on software defined networking with multiple controllers. *Journal of Network and Computer Applications*, 103, 101-118.
- [40] John, B (2015). Implementing MPLS with Label Switching In Software-Defined Networks. Thesis paper University of Illinois at Urbana-Champaign.
- [41] Enno Rey (2016, June 1) MPLS and VPLS security [On-line]. Available  
<https://docplayer.net/2887672-Mpls-and-vpls-security.html>
- [42] E., Systems, O., Edition, S., & Communications, B. D. (2011). Foundations of Modern networkin SDN, NFV, QoE, IoT, and Cloud. In *Network* (Vol. 139, Issue 3). <https://doi.org/10.1007/11935070>

## Appendix

```
habte@habte-VirtualBox:~/ryu-test/ryu$ PYTHONPATH=. ./bin/ryu run --observe-links ryu/app/gui_topology/gui_topology.py
loading app ryu/app/gui_topology/gui_topology.py
loading app ryu.app.rest_topology
loading app ryu.app.ws_topology
loading app ryu.app.ofctl_rest
loading app ryu.controller.ofp_handler
creating context wsgi
instantiating app None of Switches
creating context switches
instantiating app None of DPSet
creating context dpset
instantiating app ryu/app/gui_topology/gui_topology.py of GUIServerApp
instantiating app ryu.app.rest_topology of TopologyAPI
instantiating app ryu.app.ws_topology of WebSocketTopology
instantiating app ryu.app.ofctl_rest of RestStatsApt
instantiating app ryu.controller.ofp_handler of OFPHandler
(2898) wsgi starting up on http://0.0.0.0:8080
```

Figure 1: Starting Web topology

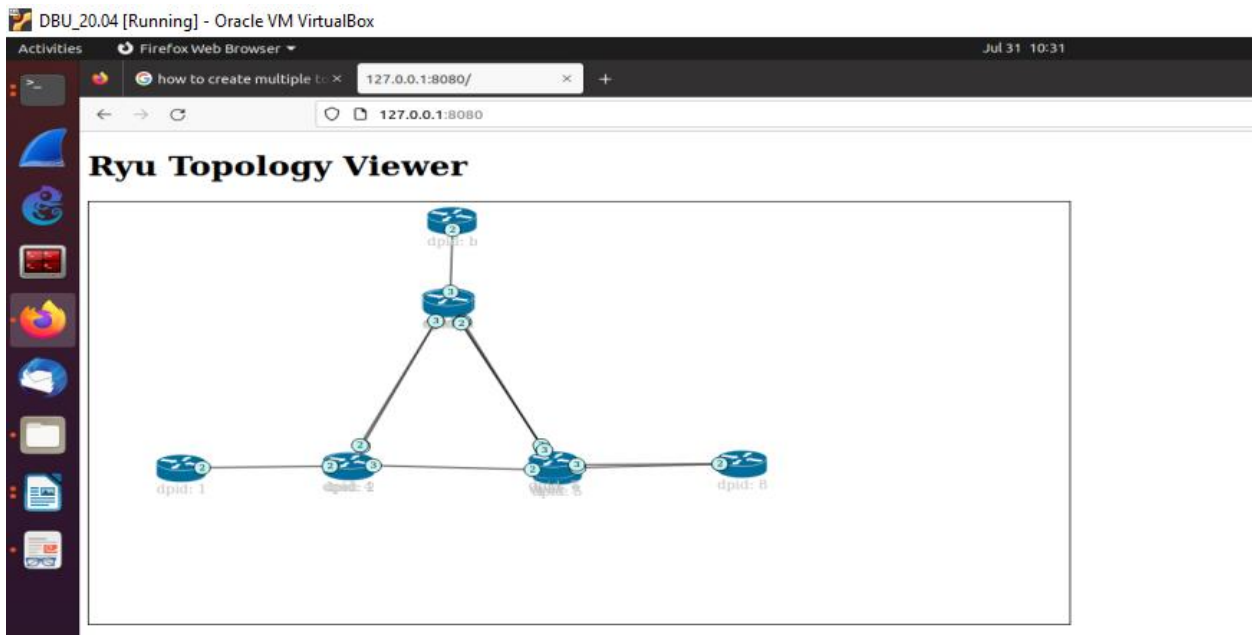


Figure 2: Topology Web access Ryu controller

```
habte@habte-VirtualBox:~/ryu-test/ryu$ PYTHONPATH=. ./bin/ryu-manager ryu/app/simple_switch.py
loading app ryu/app/simple_switch.py
loading app ryu.controller.ofp_handler
instantiating app ryu/app/simple_switch.py of SimpleSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Figure 3: Starting Ryu Controller

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet, ethernet, mpls
class MPLSHandler(app_manager.RyuApp):
```

```

def __init__(self, *args, **kwargs):
    super(MPLSHandler, self).__init__(*args, **kwargs)
    self.waiting_queue = {
}

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocol(ethernet.ethernet)
    if eth.ethertype == ethernet.ETH_TYPE_MPLS:
        mpls_pkt = pkt.get_protocol(mpls.mpls)
        if mpls_pkt:
            label = mpls_pkt.label
            self.handle_mpls_packet(label, pkt)
def handle_mpls_packet(self, label, pkt):
    if self.waiting_queue:
        # Check if there are packets waiting
        waiting_label = next(iter(self.waiting_queue))
        if label > waiting_label:
            # Forward label1 to the destination
            self.forward_packet(pkt, f"Forwarding label {label} to the destination")
            # Add label2 to waiting
            self.waiting_queue[label] = pkt
            return
        else:
            # Forward label2 to the destination
            self.forward_packet(self.waiting_queue[waiting_label], f"Forwarding label
{waiting_label} to the destination")
    # Clear waiting queue
    self.waiting_queue = {}

```

```

        self.forward_packet(pkt, f"No packets waiting. Forwarding label {label} to the
destination")
    def forward_packet(self, pkt, action):
        # This function simulates forwarding the packet
        print(action)
        # Implement the actual forwarding logic here
def main():
    app_manager.instantiate(MPLSHandler)
}
}

```

### ➤ MPLS network Configuration

=====

```

P#show running-config
Building configuration...
Current configuration : 1787 bytes
!
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption

hostname P
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
memory-size iomem 5
no ip icmp rate-limit unreachable
ip cef
!
!
!
!

```

```
no ip domain lookup
!
!
!

!
!
!
ip tcp synwait-time 5
!
!
!
interface Loopback0
 ip address 3.3.3.3 255.255.255.255
!
interface FastEthernet0/0
 ip address 192.168.23.3 255.255.255.0
 duplex auto
 speed auto
 mpls ip
!
interface FastEthernet0/1
 ip address 192.168.34.3 255.255.255.0
 duplex auto
 speed auto
 mpls ip
!
interface FastEthernet1/0
!
interface FastEthernet1/1
 no switchport
 ip address 192.168.20.3 255.255.255.0
!
interface FastEthernet1/2
!
interface FastEthernet1/3
!
interface FastEthernet1/4
!
interface FastEthernet1/5
!
```

```
interface FastEthernet1/6
!
interface FastEthernet1/7
!
interface FastEthernet1/8
!
interface FastEthernet1/9
!
interface FastEthernet1/10
!
interface FastEthernet1/11
!
interface FastEthernet1/12
!
interface FastEthernet1/13
!
interface FastEthernet1/14
!
interface FastEthernet1/15
!
interface Vlan1
  no ip address
!
router ospf 1
  mpls ldp autoconfig
  log-adjacency-changes
  network 2.2.2.2 0.0.0.0 area 0
  network 3.3.3.3 0.0.0.0 area 0
  network 192.168.10.0 0.0.0.255 area 0
  network 192.168.11.0 0.0.0.255 area 0
  network 192.168.12.0 0.0.0.255 area 0
  network 192.168.23.0 0.0.0.255 area 0
  network 192.168.34.0 0.0.0.255 area 0
  network 192.168.45.0 0.0.0.255 area 0

!
!
!
no ip http server
no ip http secure-server
!
```

```
no cdp log mismatch duplex
!  
!  
!  
control-plane  
!  
!  
!  
!  
line con 0  
  exec-timeout 0 0  
  privilege level 15  
  logging synchronous  
line aux 0  
  exec-timeout 0 0  
  privilege level 15  
  logging synchronous  
line vty 0 4  
  login  
!  
!  
end
```