ISMM: An Improved Switch Migration Method Based Load Balancing for Multiple Controllers in Software Defined Network

Muktar Abdella Jiru

A Thesis Submitted to the Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

October, 2021

Adama, Ethiopia

ISMM: An Improved Switch Migration Method Based Load Balancing for Multiple Controllers in Software Defined Network

By:

Muktar Abdella Jiru

Advisor:

Dr. Ketema Adere

A Thesis Submitted to the Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

October, 2021

Adama, Ethiopia

DECLARATION

I hereby declare that this Master Thesis entitled **"ISMM: An Improved Switch Migration Method Based Load Balancing for Multiple Controllers in Software Defined Network"** is my original work. That is, it has not been submitted for the award of any academic degree, diploma, or certificate in any other university. All sources of materials that are used for this thesis have been duly acknowledged through citation

Muktar Abdella	_____	_____
Name of the Student	Signature	Date

RECOMMENDATION

I, the advisor of this thesis, hereby certify that I have read the revised version of the thesis entitled **"ISMM: An Improved Switch Migration Method Based Load Balancing for Multiple Controllers in Software Defined Network"** prepared under my guidance by **Muktar Abdella** submitted in partial fulfillment of the requirements for the degree of Masters of Science in **Computer Science and Engineering**. Therefore, I recommend the submission of the revised version of the thesis to the department following the applicable procedures.

Dr. Ketema Adere	_____	_____
Major Advisor	Signature	Date

APPROVAL PAGE

I, the advisors of the thesis entitled **"ISMM: An Improved Switch Migration Method Based Load Balancing for Multiple Controllers in Software Defined Network"** and developed by **Muktar Abdella,** hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

Dr. Ketema Adere	_____	_____
Major Advisor	Signature	Date

We, the undersigned, members of the Board of Examiners of the thesis by **Muktar Abdella** have read and evaluated the thesis entitled **"ISMM: Improved Switch Migration Method Based Load Balancing for Multiple Controllers in Software Defined Network"** and examined the candidate during the open defense. This is, therefore, to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master of Science in Computer Science and Engineering.

_____	_____	_____
Internal Examiner	Signature	Date
_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
External Examiner	Signature	Date

Finally, approval and acceptance of the thesis is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

_____	_____	_____
Department Head	Signature	Date
_____	_____	_____
School Dean	Signature	Date
_____	_____	_____
Office of Postgraduate Studies, Dean	Signature	Date

ACKNOWLEDGMENT

Above all, I give special thanks to the almighty Allah who give me health and helped me throughout all my success. I want to express my special thanks and gratitude to my advisor **Dr. Ketema Adere** for his endless support, guidance, comment, and suggestion during the thesis work. He is a great person and one of the best mentors, always am thankful to him.

My sincere thanks also go to all my family for their support and advice in my life journey. My mom **Medina Gemechu** and My father **Abdella Jiru** your support and advice shaped me into the person I am today,"Naaf Jiraadha". Dear all my brothers and sisters, "Isin Jaalladha". Finally, I wish to thank all my friends who support me in the completion of my thesis and All Network Science and Security SIG students for their significant contribution, support, comment, and motivation during my work. Especially thanks to Anteneh Tilaye for his support, comments and suggestion in my work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Program Interface
BalCon	Balanced Controller
BalConPlus	Balanced Controller Plus
CAPEX	Capital Expense
CV	Cluster Vector
CDPI	Control to Data Plane Interface
CAMD	Controller Adaption and Migration Decision
COLBAS	Cooperative Load Balancing Scheme for Hierarchical Controller Implementation
DALB	Dynamic and Adaptive Load Balancing
ElastiCon	Elastic Controller
HA	High Availability
ISMM	Improved Switch Migration Method
ICT	Information Communication Technology
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISP	Internet Service Provider
LB	Load Balancing
NFV	Network Function Virtualization
NOS	Network Operating System
NS3	Network Simulator Version 3
NBI	North Bound Interface
ODL	OpenDayLight
OF	OpenFlow
OFNet	OpenFlow Network
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OPEX	Operational Expenses
POCO	Pareto-based Optimal Controller Placement
QoS	Quality of Service
SAR-LB	Switch-Aware Reinforcement Learning Load Balancing

SPoF	Single Point of Failure
SDN	Software Defined Network
SDNR	Software Defined Network Research Group
SBI	South Bound Interface
SSMS	Staged Switch Migration Strategy
SMDM	Switch Migration Decision Mechanism
WAN	Wide Area Network

ABSTRACT

Software-Defined Network (SDN) is a new paradigm of computer networking that is based on the concept of separating the control plane and data plane to simplify network management through central network programmability and rapid innovation. Deployment of Multiple controllers in SDN is a promising way to achieve the reliability and scalability of the SDN controller. However, it brings a new problem of load imbalance between multiple controllers as the network traffic dynamically changed and unbalanced incoming load distribution between the controllers. Unbalanced load distribution between the controllers will reduce the controller performance by increasing response time, increasing packet loss, and reducing the throughput. The existing Staged Switch Migration Strategy (SSMS), and Controller Adaptive and Migration Decision (CAMD) are two methods that were recently developed to solve such problems in which CAMD achieves better results over SSMS. However, when the incoming traffic load was elephant flow still both existing methods are not effective, this reduces the overall system performance. To solve these problems Improved Switch Migration Method (ISMM) has been proposed in our study. The proposed ISMM was used on each controller during controller load balance to decide a set of underloaded and overloaded controllers. The proposed method balances the controller load by migrating the switch from an overloaded to an underloaded controller. The implementation was simulated using the Mininet simulation tool. The performance of the proposed solution has been evaluated by using throughput, response time, and packet loss metrics and compared with SSMS and CAMD. Accordingly, the simulation results indicated that ISMM improves throughput by 7.6% over SSMS and by 1.8% over CAMD, improves response time by 8.3% over SSMS and by 4% over CAMD, and improves the packet loss by 8% over SSMS and by 1.3% over CAMD when the incoming traffic load is between 501pps and 5010pps. This study concludes that ISMM indicates a better improvement in all assessed metrics that shows efficient results over SSMS and CAMD by balancing the load between controllers.

Keywords: *SDN, ISMM, CAMD, SSMS, Elephant Flow, Multiple Controller, Controller load imbalance*

CHAPTER ONE

1. INTRODUCTION

1.1. Background of the Study

In the traditional network systems, both the control and data plane are coupled together in a single network device such as a router, switch, and the whole structure is highly decentralized. The control plane controls the configuration and programming of paths through which it controls information for data flow and forwarding. But, since this traditional network mechanism is static, once information is defined it cannot be changed till experts did another configuration on the networks. This problem makes the traditional networking systems less reactive towards the dynamic changing of traffic with the rapid growth in technological advances such as cloud computing, the internet of things, and big data that have a great impact on the amount of traffic on the internet (Belgaum et al., 2020). Traditional network technology has limitations in terms of new technological innovations, complex configuration management, time-consuming, error-prone devices, and operating costs.

To overcome the limitation of traditional networks, the idea of Software Defined Network(SDN) has emerged that is the new network architecture to enhance the traditional network, centralized management, reduced hardware costs both CAPEX and OPEX, a highly scalable, vendor-neutral that build to enable operators, error-free, flexible management of networks, dynamically configurable network that easily adapts to changing network requirements(Chandra et al., n.d.)(Hakiri et al., 2014; Jammal et al., 2014). SDN is a new paradigm of a network that separates the control plane from the data plane for simpler and flexible management of the network. By separating the two planes i.e. control plane and data plane and integrating intelligence into the SDN controller, SDN makes networks programmable via Application Programming Interfaces (APIs)(Tiwari et al., 2019).

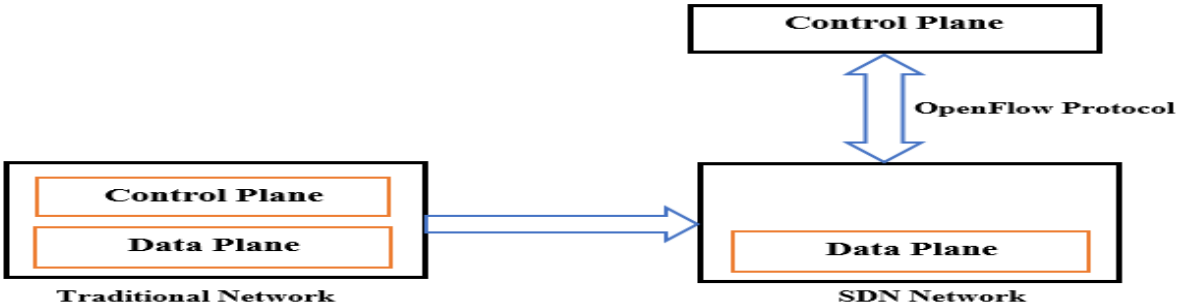


Figure 1.1 Traditional Network vs SDN

As shown in figure 1.1, a traditional network embedded together both control and data planes in the same network device and SDN separate two planes and use the OpenFlow protocol for communication between the control and data planes. SDN operates in a way that they communicate with the devices using OpenFlow protocol and this protocol states the network-related regulations to them. As all activities depend on the central controller, if one controller fails, all switches will fail, and no longer smoothly provision of services occur. As a switch forwards data for the first time, this switch finds the respective flow entry from its flow table in their memory according to which they conduct the packet and when a packet enters a switch. If the corresponding flow entry is no regulation in the table for the received packet, it will ask its controller for the latest flow table and send a packet-in request and add necessary regulations in the switch flow table after doing the necessary calculations.

SDN contains two types of the controller in its architecture i.e. single and multiple controllers. Single controller architecture contains only one centralized controller in its architecture. So, it has problems such as single point of failure (SPoF), which cannot handle a huge number of requests from switches because of limited controller capacity. To overcome those problems several works propose multiple controllers in SDN. Multiple controller architecture comes into existence to provide a reliable, scalable, and available control plane for the network. Finally, we discover that when comparing a single controller with multiple controller architecture, multiple controllers can efficiently improve the performance of the SDN networks. Therefore, in this study, multiple controllers are used because this architecture brings a great solution for the network scalability, reliability, and availability, but the problem of controller load imbalance among the multiple controllers is raised that was solved in our studies.

In this controller architecture, several controllers are used to act as logic controllers and manage the network together. If the network traffic sent to the controllers is unbalanced, it could cause overload in some controllers and others would become underloaded (Hu, Guo, et al., 2018a). This will dramatically maximize the response time of the overloaded controllers, minimize the throughput of the controller, maximize the number of packet losses, and letting some controllers unused.

The other problem of multiple controller designs is their static connection between switches and controllers which makes them unable to adapt to traffic variation in the control plane. Because this static mapping can cause load fluctuations between the controllers real networks

may vary at a different time of the day or even in a short time scale (temporal dimensions) and traffic varies at different locations of the network(spatial dimensions).

This problem may result in an imbalance in the controllers that make some controllers overloaded and others underloaded. There are two solutions to this problem. The first solution to solve all these problems, dynamically connect between switches and controllers, and the second solution in case of load imbalance, migrate the switches from more loaded to underloaded controllers to attain load balance (Kaur et al., 2018)(Killi & Rao, 2019). In general, the goals of controller load balancing among multiple controllers are to increase the quality of service (QoS), improve network performance, maximize throughput, avoid congestion, minimize response time, and minimize packet loss (Jadhav et al., 2020; Kawale et al., 2021).

In this thesis, an improved switch migration method (ISMM) was proposed based on dynamic switch migration to realize load balance between multiple controllers. The proposed method includes four modules including the load measurement and judgment module, the switch selection module, the controller selection module, and the switch migration module. Consequently, a threshold value of the controller is used to identify load imbalance in the controller, which fluctuates based on the network load over time. This means that the network adapts to the load condition. If the controller load surpasses the defined threshold, it will migrate switch to reduce its load (Ma et al., 2017).

1.2. Motivation of the Study

The problems of the traditional network are solved by the transition of the traditional network to SDN, it still has several challenges. To deploy SDN, the challenging problem is load balancing between multiple controllers. With the growth of network traffic load and the network scale, a single centralized controller can’t able to balance the load in the controllers. Therefore, multiple controllers are deployed to solve these issues of the control plane. Still, the existing multiple controller's deployments are static which makes the controller unable to adapt to time-varying traffic that results in load imbalance among controllers.

If the incoming traffic load of the controller was unbalanced, it would cause overload in some controllers while others are underloaded. This undesirable situation will maximize the response time, minimize throughput, maximize the number of packet losses, and eventually degrade the

control plane performance (Saraswat et al., 2019). This indicates that there is a necessity for an efficient load balancing method to solve the problems. So, these issues motivate us to propose a solution that balances the load between multiple controllers.

1.3. Statement of the Problem

SDN uses multiple controllers to offer scalable, available, and reliable systems for the network. But, the problems of load imbalance among the controllers are still one of the challenging issues. Load imbalance issues severely affect the performance of an entire network. So, in current years, the problem of load balancing in SDN multiple controllers has become an interesting research area. In multiple controllers, if the incoming traffic load to the controller was not balanced between the controllers, it would cause overloading in some controllers and some others would remain underloaded (Yang Zhou et al., 2018).

The controller load imbalance problem in SDN multiple controllers occurs when uneven load distribution among controllers because of the static configuration between switch and controller. When mapping among switch and controller is static, it cannot adapt for time-varying traffic load since traffic conditions fluctuate at different times of the day or even on a smaller time (temporal) and traffic fluctuates at different locations in the network (spatial) that result in load imbalance problems among controllers, thereby overloading some controllers while under loading some others. The existing dynamic switch migration has the problems of the method to select switch and select controller form migration to improve network performance and support only small scale networks(Hu, Yi, et al., 2018).

The unbalanced distribution of incoming load on the controllers will maximize response time for processing flows, increase the number of packet losses, and reduce the controllers' throughput. So, there is a necessity for an efficient load balancing technique that solves the issues above and improves system performance than existing methods. By taking this into account, two recently developed SSMS and CAMD methods are tried to solve the problem of controller load imbalance. However, both methods were not effective when the incoming traffic load is high this leads to a performance reduction in the overall system. The above issues are solved by the proposed ISMM which is simulated using the Mininet simulation tool and ODL controller. Finally, the switch is dynamically mapped to the controller and the switch migrated from overloaded to the underloaded controller to achieve a balanced load(Sahoo & Sahoo, 2019)(Hu et al., 2017).

1.4. Research Questions

This thesis intends to answer the following four research questions:

RQ1: How to study the effect of load imbalance on the network performance in SDN?

RQ2: How to dynamically map the connection between switches and controllers?

RQ3: How to improve existing methods for balancing load between controllers?

RQ4*:* How to evaluate the performance of the newly proposed solution with existing works?

1.5. Objectives of the Study

1.5.1. General Objective

The general objective of this research work is to design an improved switch migration method based on load balancing for multiple controllers in a software-defined network.

1.5.2. Specific Objectives

To achieve our general objective, the following specific objectives are addressed throughout the study.

- To study the effect of controller load imbalance on the network performance in SDN.
- To build the method that dynamically maps switches and controllers to balance the load.
- To improve existing methods for load balancing between multiple controllers.
- To evaluate the performance of the proposed method with SSMS and CAMD works.

1.6. Methodology of the Study

To address the above objectives and to answer the research questions, the following methods and techniques are used:

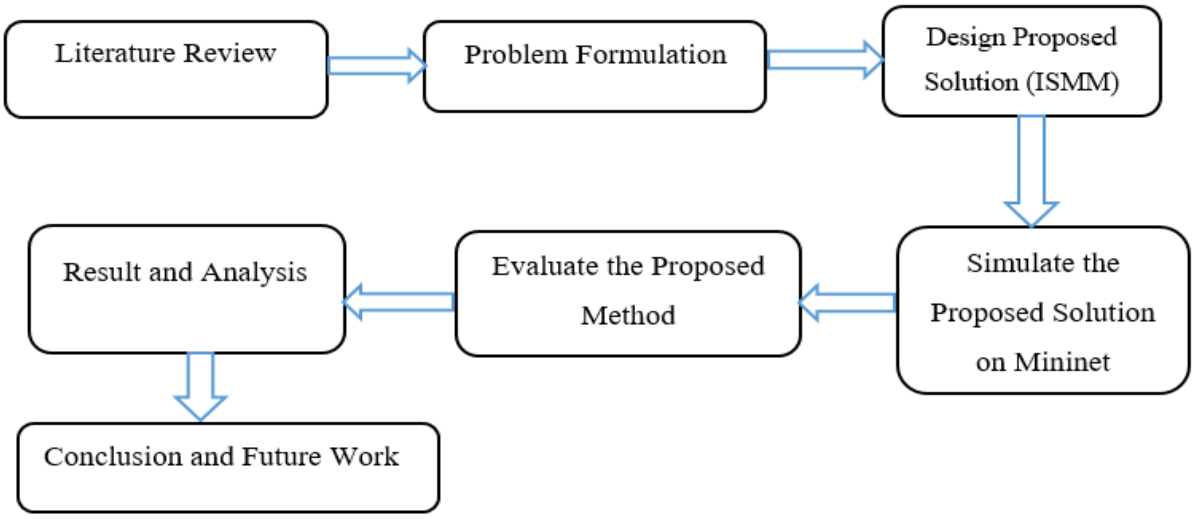


Figure 1.2 Procedure of the Study

1.6.1. Literature Review

To achieve research objectives several works such as research papers, magazines, conference papers, guideline manuals, and journal articles are reviewed to understand the research work and to gain a deeper understanding of the research domain.

1.6.2. Problem Formulation

After reviewing different recent articles and papers the problems are formulated, then the problem formulated was again and again reviewed.

1.6.3. Design Proposed Solution

The new method was developed based on the switch migration which is used for modifying the controller load distribution by migrating the switch from overloaded controller to underloaded controller. After migration of the switch from the initial controller to the target controller, all packet-in messages generated by the switch will be received and processed by the target controller and the role of the controller will change according to OpenFlow v1.3.

1.6.4. Simulate the Proposed Solution

The hardware and software tools used to simulate the proposed solution are shown in table 1.1.

Table 1.1 Software and Hardware Tools

Software	***Hardware***
• Operating System: Ubuntu 20.04.2 LTS and Microsoft Window 10. • Simulation Tool: Mininet 2.3.0 Simulator • Programming Language: Python • Controller: OpenDayLight (ODL). • OpenFlow Version 1.3. • Draw.io: To draw the architecture and diagram in the study.	• RAM: 4.00GB. • System Type: 64-bit. • Hard Disk: 1.0 TB. • Processors: Intel® Core™ i5-6501 CPU @ 3.20 GHz x 4 • Graphics: Mesa Intel® HD Graphics 530(SKL GT2).

1.6.5. Evaluating the Proposed Solution

The proposed ISMM is compared with existing SSMS and CAMD on jupyter notebook editor and Microsoft Excel 2016 to plot the comparison result. Lastly, the results gained from this simulation are discussed clearly and reported. The proposed solution can be evaluated based on the performance evaluation metrics such as response time, throughput, and packet loss(Mamushiane et al., 2018).

1.6.6. Result Validation

The simulation result was obtained from the implementation directory of Mininet Simulator. The results were described with a table and graph using Excel by the parameters listed above.

1.7. Scope and Limitation of the Study

1.7.1. Scope of the Study

As the research is constrained with time, we focused only on the switch migration method to solve the problems. Balancing the controller load is a significant task in our thesis that enhances the network performance. The proposed method solves the problems of load imbalance between multiple controllers and improves the existing works. Finally, the proposed solution was compared with the existing method SSMS and CAMD that were raised in related works. Performance assessment metrics such as packet loss, throughput, and response time are used to evaluate the proposed solution and compare results with existing methods. The simulation used four controllers and twelve switches.

1.7.2. Limitation of the Study

A lot of issues were raised to solve the problem of load imbalance and static connection between switch and controller. This research, will not consider the real deployment of the network, the controller security issues, controller failure scenarios, and migration costs due to time constraints. Because these problems need another further study.

1.8. Significances and Beneficiaries of the Research

1.8.1. Significance of the Research

The balanced load distribution between multiple controllers in SDN can provide the following significant benefits:

- Improves response time, throughput, and overall network performance.
- Easy management of all network devices because of its centralized management and can be configured without no human interaction.
- Reduce the costs for different organizations.
- Reduce complexity.
- Balance load among controllers to reduce load imbalance.
- Encourages innovation with common and well-defined APIs.
- It will create awareness for researchers who want to spread their study on load balancing.

1.8.2. Beneficiaries of the Research

The beneficiaries of the research are the following:

- ***Network Administrators:*** Network administrators uses this investigation of the thesis for managing the network operation easily.
- ***Telecommunication***: Telecom organization uses the result of the thesis to deploy the SDN and to spread the organization's study further.
- ***Network Experts***: It is used by networking organizations, network experts, SDN operators, distributed controller developers, and testers in developing a new architecture.
- ***Future Academicians and Researchers:*** academicians and researchers can be used this thesis as references for further studies. It is also used at different research institutions, enterprise networks, and WANs as an academic reference.

1.9. Application of the Study

The result of this work contributes to balancing controller loads for multiple controllers in SDN. This study addresses a load imbalance impact on controllers. The simulation would be performed by comparing SSMS and CAMD with the proposed ISMM using the Mininet simulation tool. Consequently, our study balances the controller loads by migrating the switch from overloaded to underload. Different companies such as Facebook, Yahoo, Microsoft, IBM, Cisco, and Google use SDN. This work applied to one of the following areas such as large-scale networks, campus networks, home networks, cloud data centers, internet service providers, enterprise networks, wide area networks, and telecommunication.

1.10. Organization of the Thesis work

This thesis is organized generally into five chapters as follows:

- Chapter one describes the introduction, motivation of the study, statement of the problem, objectives, methodology, scope, and limitation, the significance of the research, and beneficiaries of the research, and application of the study.
- Chapter two explains literature reviews and related works mainly help the reader to know more about the areas and the specific issues which issued solved by the thesis.
- Chapters three describe the proposed ISMM and the general design of the proposed solution.
- Chapter four describes the simulation study and simulation results.
- Finally, chapter five describes the conclusion, output of the study, and future works.

CHAPTER TWO

2. LITERATURE REVIEW AND RELATED WORK

2.1. Overview of the Traditional Network

The internet has led to the development of a digital society in which almost everything is networked and accessible from anywhere. The Internet has grown rapidly and has been used for a variety of real-world applications such as web services, database access, real-time audiovisual communications, and broadcasting. However, despite their widespread use, traditional IP networks are very difficult to manage and still complex. It is challenging to configure the network according to pre-defined policies and to reconfigure and also leads to load imbalance between network devices(Kreutz et al., 2015).

In traditional networking architecture, the transport network protocols and distributed control running inside the switches and routers are the important technologies that allow information in the form of digital to travel from place to place over the world. Even with their widespread implementation, traditional IP networks are hard to manage and complex. To express the preferred high-level network policies the network operators need to configure each network device separately using low-level and often vendor-specific commands. In addition to the configuration complexity, network environments have to suffer the dynamic of errors and adapt to load changes. Applying the essential rules in such a dynamic situation is therefore extremely challenging (Benzekki et al., 2016; Vasudevan & Nayak, 2018).

Dynamicity and automatic control of a network without manual intervention is the challenging task of any network architecture, although it is useful and creative. This automatic configuration of devices and management of different activities is poor in the traditional network. In this, network devices are integrated virtually, and the network traffic is controlled by controlling elements and forwarded by the forwarding elements in the form of digital packets. The data plane and the control plane in the traditional network are embedded together inside the networking devices, delay innovation, reduce flexibility, and develop the networking infrastructure (Lin & Zhang, 2016). In a traditional network, their two planes are tied up together, thus leading to a rigid and closed design, as you can see in figure 2.1 that adapted from (Eissa et al., 2019).

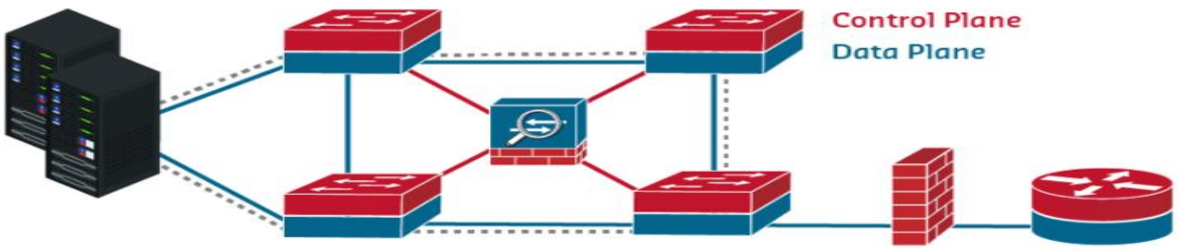


Figure 2.1 Architecture of Traditional Network

New policies are defined in the management plane and applied by the control plane. The data plane helps with traffic flow by executing these policies as directed by the control plane. The problem with traditional networks is when managing modern data network traffic that they have issues of hard and complex to manage modern data networks, deploying new networking features is very expensive, inefficient way traffic handling, difficult to handle network configurations and errors(Kreutz et al., 2015). To overcome the challenges of the traditional network architecture and the technological improvement of the network, the SDN paradigm is adopted to eliminate the maintenance processes of the network infrastructure and ensure easy management. The SDN is a new paradigm of a network architecture that separates two planes to overcome the limitations of traditional networks and offers many opportunities for innovation. Figure 2.2 adapted from(Eissa et al., 2019), shows three planes of SDN architecture.

- Control plane: consists of SDN controller and network operating system.
- Data plane: consists of the network devices.
- Application plane: consists of network business applications.
- Southbound API and Northbound API for communication between planes.

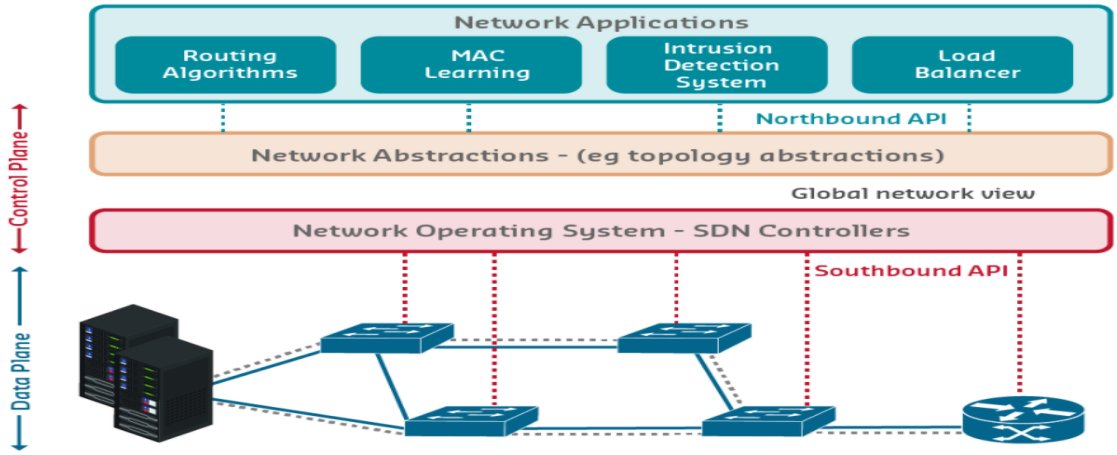


Figure 2.2 Architecture of SDN

SDN is a type of programmable network that gives a solution to traditional networks by different perspectives such as availability, management of the network, maintenance cost, utilization of controller resources, more agile, reduces complexity, encourages innovation, and programmable. The core clue of SDN is to implement a programmable network by separation of control plane and data plane to improve network performance. With the separating of these two planes, all network management task is centrally controlled by an SDN controller which is NOS, and data plane devices are responsible for the forwarding of the traffic. The main issue in SDN is the load balancing between multiple controllers, which gives an overview of the whole network (Neghabi et al., 2018). In the next section, the SDN overview, the architecture of the SDN, and more specific emphasis on the core issue of the research are discussed.

2.2. Software Defined Networking

2.2.1. Overview of the SDN

As an ongoing and new network paradigm that solves challenges found in traditional IP-based networks, SDN comes into existence as a promising method for future network technologies. This new network paradigm separates the control plane from the data plane and logically centralized control by enabling programmable network devices. This decoupling of layers network allows for a separate controller entity that changes the data forwarding rules in OpenFlow switches and enables SDN to react kindly to the change network demands. With this separation, forwarding devices can be applied in a logically centralized controller. The forwarding tables of switches that configure logically centralized controllers are responsible for network configuration, policy enforcement, topology management, flow table entry, the discovery of the link, and forwarding of the packets of communication flows. SDN is an advanced way of designing, implementing, and handling networks by separating the network controller and forwarding devices for achieving an enhanced user experience. This network separation offers numerous advantages such as reduced complexity, controllability, flexibility, rapid innovation, and easier management to the future internet (Eissa et al., 2019)(Hamdan et al., 2021)(Sharma, 2021).

In the SDN, since the controller has a global view of the network, it can simply add and delete rules dynamically. It is also capable to drop a harmful attack that affects the network performance. This new network can reconfigure the network-based condition from network operators which can add and remove packet forwarding rules. Only the first packet of each new

flow arrives at the controller. Based on this packet, the controller defines the rules for forwarding devices such as routers and switches. The next incoming packet-in requests of this new flow check the flow table entries and forward the packet to the next forwarding devices. Network management decisions made by network elements at the data plane become easy because the controller only processes information related to the logical topology of the data plane network traffic and load balancing that manages the network traffic based on the configuration specified in the control plane(Benzekki et al., 2016; Sharma, 2021).

In SDN network architecture two planes are communicated through an OpenFlow protocol. Devices that support OpenFlow contain two components. I.e. flow table that states how packets are processed and forwarded in the network and an OpenFlow API that handles the exchange between switch and controller. One of the most important benefits of SDN is its intelligence of the network which is centralized in controllers that is software-based in which the network looks to applications that highly explain network design and operations. The centralized control plane enables optimization of the network resources and network management in which the logical centralization of SDN can be attained by a single controller or by multiple controllers.

2.2.2. Architecture of SDN

SDN is a computer network that is implemented, managed, and produced using the software. It is an architecture that is used to decouple the data plane and control plane. Besides, it is fast becoming the preferred solution for those who are struggling to overcome the limitations of traditional networks and determine where to send traffic. The data plane comprises the network devices, and the controller acts as a brain that centralizes network intelligence. The OpenFlow interface is used for communication between two layers, in which the controller transmits different rules to the data plane based on the dynamic needs such as policies. In this communication, the data plane forwards a given packet to the next node based on the simple rules provided by the controller. This new architecture of the network allows the implementation of a programmable, scalable, and flexible network. the detailed layers of SDN architecture with its main layers are shown in figure 2.3 that adapted from (Semong et al., 2020), which is proposed by the SDN Research Group(SDNRG) (Wickboldt et al., 2015).

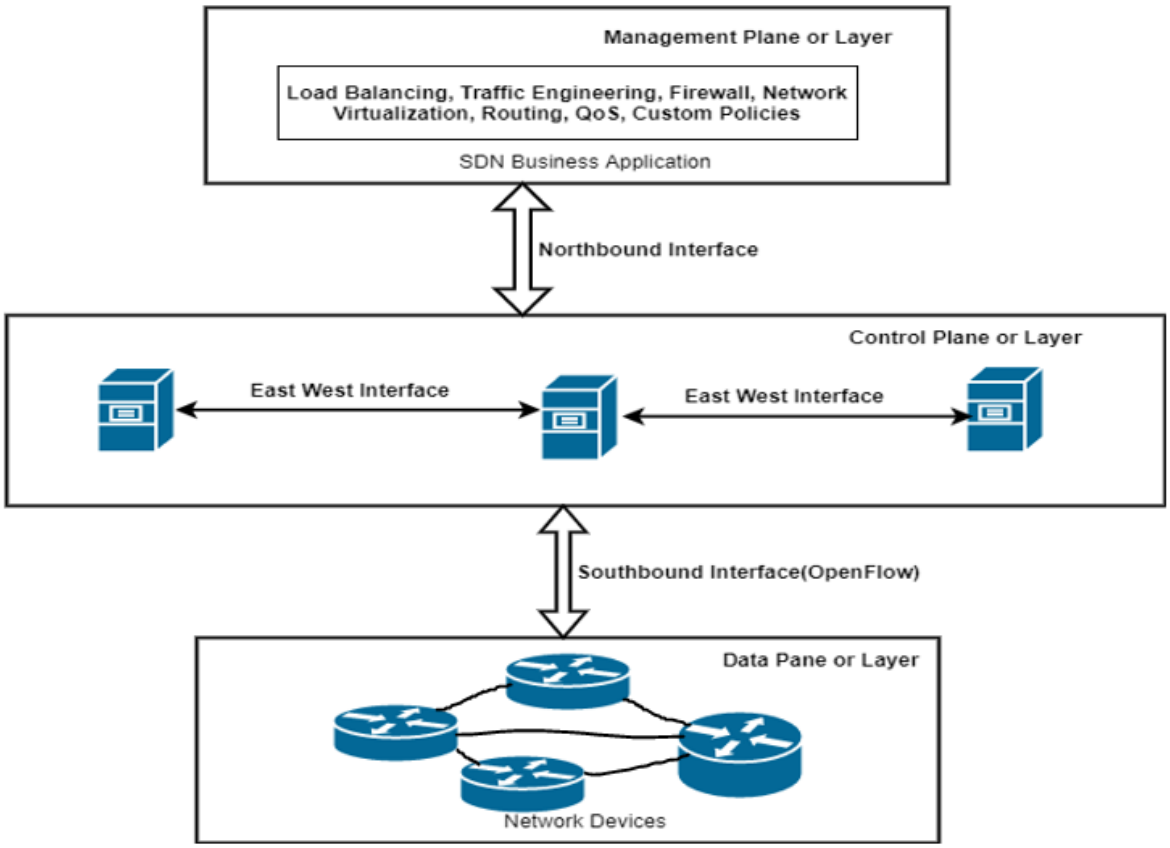


Figure 2.3 Detailed Layers of SDN Architecture

This SDN architecture has three layers called data layer, control layer, and application layer, and three interfaces like northbound interface (NBI), East-west interface, and southbound interface (SBI) as revealed in figure 2.3**.**

Management Plane: it is also called the application plane that is responsible for establishing and defining policies and rules for the network including security monitoring, load balancing, routing, network virtualization, QoS, and traffic management. The controller communicates with the application plane through NBI, in which SDN applications communicate directly, explicitly, and programmatically with the network and request of the controller.

Control plane: consists of one or more controllers that forward the command sets and policies defined by the network applications to the data plane via the southbound API interface. It acts as the brain of the network in which it has a global view of the network. It is responsible for tasks such as configuring the connection to devices at the data plane, proactively planning the movement of data, and installing forwarding rules to the data plane. In addition, it manages the creating of a network view and the availability of the data plane in the network.

Data Plane: this plane includes both virtual and physical forwarding devices such as switches and routers that are connected over a wireless or wired network. It is used to forward packets according to the route given in the flow tables for each device, and the complex method doesn't run on this plane. The flow table contains a header, a match, and an action.

Northbound Interface (NBI): it is an interface that provides the essential support for communication between the application and control plane. This interface provides commands that are matched for the SBI and do not have the standardized protocols defined for them.

East-West Interface Protocol: it is the interface that is used to manage communications between multiple controllers, especially in distributed multiple controllers which are used to increase network performance.

Southbound Interface (SBI): it is a standard and common protocol for SDN which is used between control to data plane interface (CDPI) that provides support for the communication between the control planes and data planes.

2.3. Controllers in SDN

2.3.1. Single Controller

A single controller has advantages such as the controlling of the centralized network node and the simple abstraction of the network infrastructure but has disadvantages concerning aspects such as scalability and reliability. In the early stages of SDN design, a single controller manages the entire network. In figure 2.4 that adapted from (Hu, Guo, et al., 2018b), controller (C1) manages four switches in the network. When the source host sends a new packet to the switch (S1), the switch cannot perform the forwarding function due to the lack of sending information for the new packet. Then, the S1 sends messages packet-in to the C1 to receive the sending for the new packet. C1, the switch forwards the packet to the next device. Finally, the packet successfully reaches the destination host. The controller plays a significant role in the transmission of the data traffic. However, because network traffic increases rapidly a single controller cannot receive a big number of flow requests sent by switches due to the limited capacity of the controller. Therefore, it’s necessary to deploy multiple distributed SDN controllers to overcome the limitation of being limited to a single SDN controller in which is a balanced load among controllers (Paliwal et al., 2018).

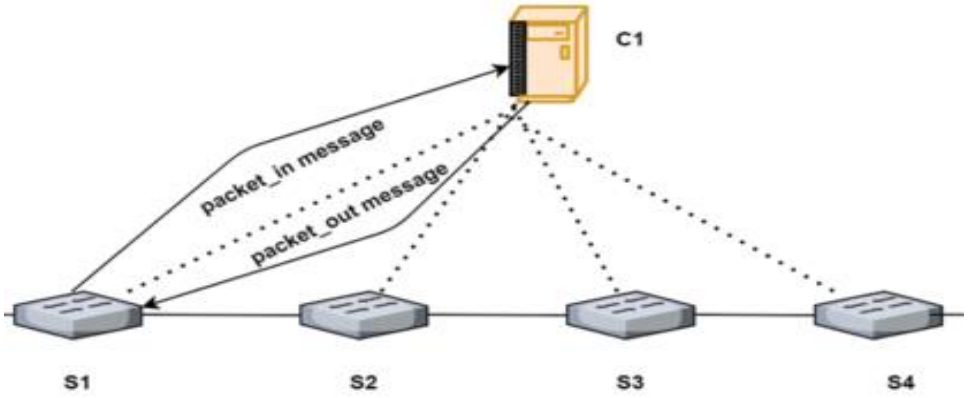


Figure 2.4 Single Controller

2.3.2. Multiple Controllers

Multiple controllers in SDN refers to SDN in which more than one controller works collaborate to improve performance, availability, reliability, and scalability. The large-scale network and heterogeneous SDN must be split into several sub-domains, and those domains require at least one controller. Compared to single controllers, the multiple distributed controllers have advantages in terms of scalability, reliability, availability, and high performance with the increased demand for requests. In figure 2.5 that adapted from (Hamdan et al., 2021), multiple controllers became a new SDN paradigm that solves the single controller problem. In the below figure, C1 and C2 share a certain logic in a logically uniform way, so that both C1 and C2 will explicitly activate sending routes on all relevant switches when new packets arrive at S1, which means that it can adequately relieve the flow processing load of a single controller. To solve problems using multiple distributed controllers is an alternative that able them to communicate through east-west interfaces that allow these controllers to interact with each other(Muluye, 2020).

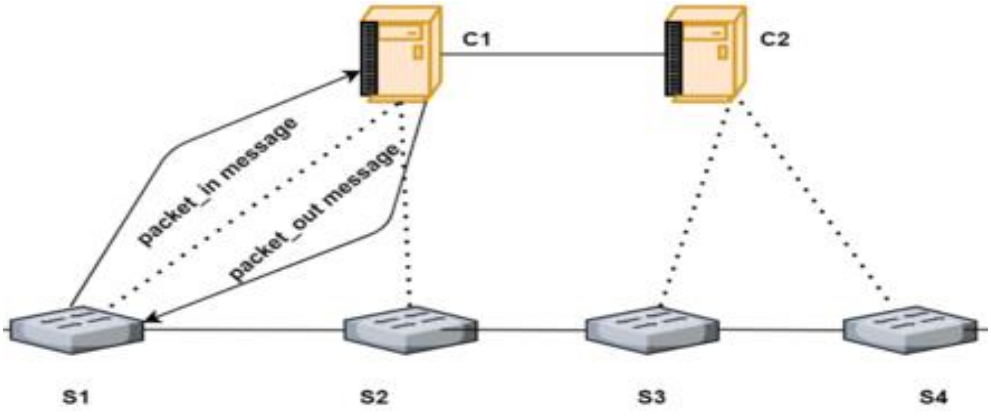


Figure 2.5 Multiple Controller

2.3.2.1. Advantages of Multiple Controllers.

Reliability: If a controller fails, the operation of the network controlled by the controller is seriously disrupted, so the flows are redirected onto new separate paths with the node. Therefore, examining the reliability of the controller node has an important effect on the multiple controllers' reliability.

Scalability: The current OpenFlow approach can result in a single controller overload which decreases network scalability. But, the multi-controller architecture is an applied solution for allowing controllers to be added and removed dynamically and requests can be balanced load to solve SPoF which is the merit of the single controller.

Administration: The single controller ability is limited with the contrast to multiple controllers, mainly in the management of the large-scale network. The cooperation of multiple controllers increases the managing efficiency for multi-domain and heterogeneous networks due to their different features.

Latency reduction: If a single controller responds to all events in sequence, it will overload the controller and increase latency. To answer this difficulty, it is better to use multiple controllers where you distribute the load between the controllers and reduce latency.

Robustness: Since the multiple SDN architecture implements multiple back-ups, it avoids a SPoF and also increases the network robustness. When one controller fails others manage switches related to it and load balancing takes place to maintain network stability, performance and also reduce the network failures which occur due to controllers' failure.

2.3.2.2. The architecture of Multiple SDN Controller

Multiple controller architectures use more than one controller in networks to address the challenges faced by a single controller network. It is used to improve controllers' scalability and reliability. Load balancing between multiple controllers is another issue to consider (H. Wang et al., 2018). There are two categories of multiple controllers, which are described in the sections below.

2.3.2.2.1. Hierarchical Architecture

In the hierarchical controller, each controller is located at different layers which are classified into the root and local controllers. Local controllers are near to switches and accountable for the network in their domain, and root controllers are accountable for maintaining the whole

network information. Figure 2.6 depicts the architecture of the hierarchical controller. However, it has become impossible for the controllers within the tiers to communicate with one another within the system, thus limiting the use of the second layer services.

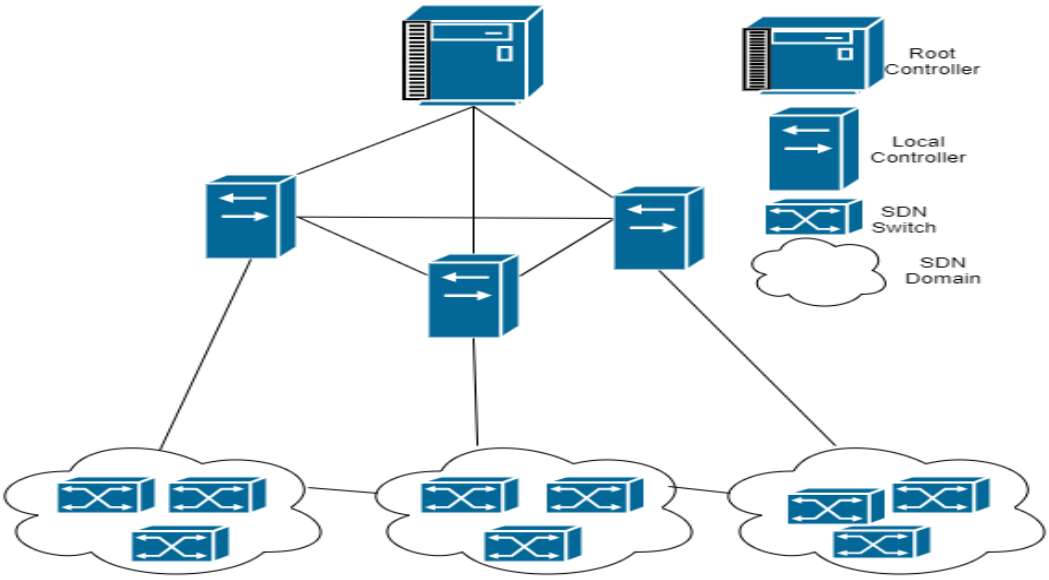


Figure 2.6 Hierarchical Architecture

2.3.2.2.2. Horizontal Architecture

In this multiple controller architecture, all controllers are placed on the same level. This is why it is also known as a flat model. Although each controller establishes different domains, all controllers are considered to be root controllers and are familiar with the complete state of the network. This architecture includes controllers such as Onix, HyperFlow, and ODL. East-west interfaces are used for communication between each controller and the management of the network state(Bannour et al., 2018). Figure 2.7 depicts horizontal architecture.

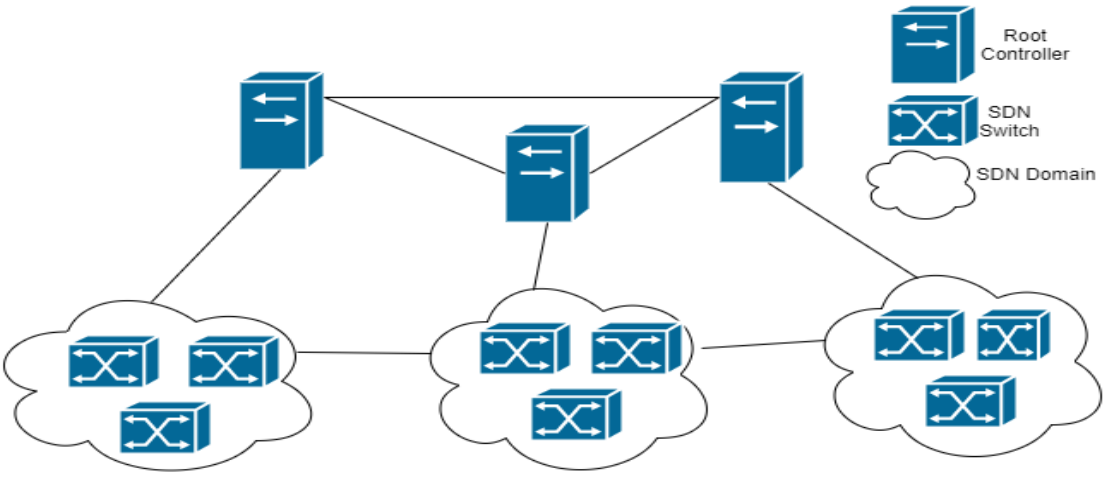


Figure 2.7 Horizontal Architecture

Since multiple controllers are better than single controllers in many ways. As we reviewed above on the benefits of multiple controllers over a single controller, this research will also focus on the multiple distributed controllers main challenges known as load imbalance, which affects the network performance.

2.3.2.3. Challenges of Multiple Controllers

As the network raises in size, the single centralized controller cannot achieve the growing request for flow processing. Therefore, the most promising solution to overcome the drawbacks of a single controller is deploying multiple controllers. But, it brings new challenges such as load balancing, consistency, controller placement, availability, security, and scheduling.

Controller load imbalance: The introduction of multiple controllers divides the network into several SDN domains, while controllers monitor local switches in the domain. However, due to the varying network traffic and the static association between switches and controllers, the controller is likely overloaded. Upon reviewing the literature, we conclude that existing research solves the problem in two ways by clustering controllers and by switch migration. Therefore, for a given multiple controller SDN network, it is essential to ensure the nice load balancing performance of multiple controllers.

Consistency: Multiple controller designs can divide the entire network into multiple domains, with each controller managing its SDN domain. To ensure that packets are transmitted correctly on the network, the controllers must interact with the domain information to ensure a consistent view. Therefore, controller consistency becomes an important issue for SDN multiple controllers as well. The multiple controllers must make a decision based on consistent and coherent network information. However, during data transfer, the synchronization between controllers and simultaneous strategic conflicts between controllers can lead to an inconsistency in the controller state.

Control placement: CPP is a main issue in multiple SDN architectures and addresses the problems such as where the controller will be placed and the number of controllers needed. While these two issues are important, Nondeterministic Polynomial (NP)-hard problems and the aspect of the location problem, such as latency reduction and load balancing are addressed.

Availability: HA is a significant feature of nowadays services that need to be available every time to a customer requests for particular service or resource and is generally stated as a

percentage of available in a given year. Implementation of server hardware, server operating systems, and redundant network components from network providers often implement backup services to provide HA.

Security: It is about protecting information from theft or damage to hardware and software, as well as service interruptions. Securing SDN involves the physical security of the hardware as well as the prevention of logical threats that could originate from the network or the data. SDN vulnerabilities are the gateway to intentional or accidental security attacks.

Table 2.1: Comparison between Single Controller and Multiple controllers

SDN Controllers	**Advantages**	**Disadvantages**
Single SDN Controllers	• used mainly if packets input is low • easy to implement	• not scalable and reliable • SPoF • Load balancing
Multiple SDN Controllers	• Reliability and scalability. • No SPoF. • enhanced control layer and better performance	• Static mapping between the switch and the controller. • Unbalanced load distribution

2.4. Load Imbalance Problem in Multiple Controller

Multiple controllers are implemented to improve the problems of the single centralized controller. However, existing implementation with multiple controllers represents the control statically. This means the relationships between switches and controllers that cannot adapt to time-varying traffic, which generates a load imbalance between controllers. This undesirable situation maximizes the response time of the controllers to process requests from the switches, increases controller throughput, and increases the number of packet loss which ultimately reduces the performance of the controller (Yao et al., 2018).

The challenge of controller load imbalance, which is addressed in this research study, is solved through the switch migration method. An improved switch migration method is proposed as a new dynamic adjustment for the controller load to resolve a load imbalance in the multiple

controllers. Migrating switches from over-loaded controllers to under-load brings balanced distribution of load on the network, but at the same time determining which switches to migrate to which controllers while keeping a balanced load on the network is the main concern in this study. We consider the maximum threshold capacity of the controller to migrate switches while solving the problem. Finally, the balanced load is used in many companies such as data centers, WANs, Internet service providers, and telecommunications.

2.5. Method to Solve Multiple Controller Load Imbalance

Upon reviewing the different literature, we conclude that existing research solves the controller load imbalance problem in two ways, which are clustering controllers and Switch migration. However, by comparison, controller clustering pays more attention to architecture design by building a dynamic controller resource pool, while switch migration focuses on adjusting the controller load balancing to maintain efficient load balancing. So, we use switch migration to solve the problems in this research study.

2.5.1. Controller Clustering Method

In the clustering controller method, a network contains a super controller and multiple regular controllers that make up the controller resource pool. The super controller is used solely to manage all controller loads and periodically collects the number of data flows in each domain from the regular controllers. A regular controller manages its domain and regularly uploads the load information with a controller interaction system. When the traffic load increases, the super controller executes the load balancing method and assigns each switch to a specific controller. The controller clustering allows the load information to be collected centrally and the super controller performs balanced load management without other unnecessary overloads in between the regular controllers. The following method is used in controller clustering to solve the load balancing problem.

BalanceFlow: It is a controller clustering-based controller load balancing based on a hierarchical implementation. The main advantage of this method is the flexible adaptation of the flow requirements processed by each controller without introducing additional propagation latencies. All controllers in BalanceFlow retain their load information and regularly publish this information via a cross-controller communication system. If the traffic conditions change,

one of the BalanceFlow controllers is selected as the super controller, which divides the traffic and reassigns various flow settings to the corresponding controllers.

Cooperative Load Balancing: Like BalanceFlow, this technique also defines a super controller to handle the loads on the controllers. Cooperative Load Balancing Scheme for Hierarchical Controller Implementation (COLBAS), which is based on controller cooperation through communication between controllers. The main idea of this scheme is similar to BalanceFlow, but the authors use a greedy method to reallocate the loads on the controllers. In particular, COLBAS can keep the system performance high and the costs for load redistribution low.

Cluster Vector (CV): In this method, load balancing is done using a self-defined CV label, which is a vector that contains the addresses of the controllers in the same cluster, while at the same time removing the dependency between the super controller and the regular controllers. The proposed design consists of two levels including high-level operations in a super controller and low-level operations in a regular controller. Each controller has its CV, and a regular controller finds the address of other regular controllers from its CV and uses the address to consult other regular controllers about their load.

Dormant Mechanism Model: Various authors design a flat architecture-based, dormant mechanism model for multiple controllers to save network resources, reduce power consumption, and improve controller utilization. The key idea is to let some idle controllers enter the dormant state to be inactive or power off when the network load is light.

2.5.2. Switch Migration Method

The switch migration method is used to handle controller load imbalance problems. Switch migration is the method in which the load on the controllers has balanced efficiently through migration of the switches from the overloaded controller to the underloaded controller. The following switch migration methods are used to solve the controller load balancing problem.

Elastic Controller (ElastiCon): It is a dynamic multiple controller architecture method used for switch migration. The elastic method contains four modules including load measurement modules, load adjustment, decision modules, and action modules. The load measuring module detects the load from each controller and sends the load information to the load adjustment and decision module that determines the load distribution between the controllers. The action module performs control actions such as migrating a switch, adding and removing controllers to achieve dynamic control of controllers and switches. ElastiCon regularly manages the load

on each controller, identifies imbalances, and automatically balances the load between the controllers by migrating switches from an overloaded controller to a lightly loaded controller.

Game-Theoretic Method: In this method, the switch migration method uses game theory. By using underloaded controllers as game players and switches as commodities, a zero-sum game model is used to emulate the skills to migrate switches between cluttered controllers. The game model is fast and efficient to achieve switch migration but is not suitable for large networks due to the high complexity of the method design.

Distributed Decisions Scheme: in this method, the switch migration problem and a network utility expansion problem are aimed at maximizing the number of service requests under the available control resource.

Load Informing Strategy: It is a load-balancing technique based on a load information strategy for controllers that strongly builds a distributed decision architecture that contains four modules those are load measurement, load informing, switch selecting, and migration decision.

Balanced Controller (BalCon): It is a heuristic method based on two key observations. An effective switch migration must take into account the communication patterns of SDN switches, and the switch migration must be processed with cluster granularity. Switches with strong connections, those that have the shortest distance to the controller should always be assigned to the same controller. BalCon significantly reduces the number of migration changes for small-scale networks.

Table 2.2: Method to Solve Load Imbalance Summary

Method	**Advantages**	**Disadvantages**
Controller Clustering	• Collect load info centrally • Save power under dormant state	• propagation latencies • Inefficient load balances under high traffic loads.
Switch Migration	• Use different modules to solve problems • Improves controller load balance issues to some extent but is not efficient.	• Some methods of switch migration can’t accept high traffic loads. • The same is applied only on small-scale networks.

2.6. Related Works

In this section, the research area related to our works that have been done for solving the problem of controller load imbalance was realized and also the gap relevant to our work was discussed. Almost several researchers have proposed several methods, mainly including controller clustering and switch migration to balance controller load issues in the multiple controllers. The solution contains both strength and limitations, so how to evaluate their performance is very important.

The other work in (Sahoo & Sahoo, 2019), based on the CAMD method proposed was discussed that the least loaded controllers are selected to accept the load change during switch migration. The CAMD method leads to reduced network efficiency, low network performance, high response time, and high packet loss. With this method, when a controller becomes overloaded, several switches must migrate from the domain of the current controller to another domain of the unloaded controller. After the migration, the new controller will act as the master controller and the old controller as the slave controller. Generally, the CAMD method consists of four modules which load estimator, controller adaptation, switch selection, and switch migration. Is not effective when incoming load traffic is under elephant flow.

In the work in (Hu et al., 2017), the challenge of load balancing of multiple controllers is addressed. The deployment of multiple controllers introduces the load imbalance problems between multiple controllers, this happens when unequal load distribution between controllers because of the static configuration between switch and controller. To solve these issues the author improves the existing switch migration approach and proposes a stated switch migration strategy (SSMS) to balance controller loads in a multi-domain network. SSMS contains two stages. The first stage of SSMS, combined several controller overheads, selects the migration entities involved overloaded domain and underloaded domain based on genetic algorithm. In the second stage of SSMS, the sequential migrating of switches is applied through setting subdomain migration degree and switch validity. Finally, SSMS can achieve effective controller load balancing with better performance in distributed SDN networks, but only for the small-scale network. This work doesn't consider how switch and controller are selected and is also not efficient under dynamic traffic conditions if the load becomes too huge.

The work in (Yang Zhou et al., 2018), proposes an elastic switch migration model, which is based on a three-dimensional earth mover's(EMD) distance algorithm, to protect strategically main controllers from reconnaissance and saturation attacks in SDN. This system confirms

optimal allocations of switches and the minimal difference between controllers. A heuristic method is also designed to reduce the computational complexities, improving the scalability of switch migration for large-scale networks. This work significantly disguises important controllers against reconnaissance or eavesdropping by reducing the differences between controllers. Saturation attacks can also be prevented, as the switches are continuously migrated from overloaded-loaded controllers to under-loaded controllers and within the limits of capacities of controllers. This model does not work well because the load on the target controller is not taken into account in this model.

The work in (Adekoya et al., 2020), had proposed an improved switch migration decision algorithm(ISMDA) that solved the problem of load balancing when the incoming traffic flow is high. To balance the load in this method controller load status is used that determine the set of under-loaded controllers. The constructed migration model was used to identify both migration costs and load balancing variation simultaneously for the best selection of controllers from the set of underloaded controllers. Finally, this work is compared with two existing works DALB and CAMD. In this work, it is expected that future research will forecast the load of the controller in an advanced way. The method in which the switch is selected and the controller is selected is also not addressed in this work, hence it affects network performance.

The work in (Yuanhao Zhou et al., 2015), proposes the DALB method for solving the problem of load imbalance. The challenge of DALB is related to an increase in overloading when chosen the underloaded controller and the CAMD method is related to the challenge of switch selection. Selection of switch is inefficient, low throughput, high response time, and high packet loss in CAMD. During the switch migration, this method selects the nearest controller from the set of under-loaded controllers to accept the load change. In each phase of the migration, the closest neighbor controller is always taken into account for the migration, but the process leads to traffic congestion by accepting the load change.

The work in (Sahoo et al., 2020), investigates that as the need for IoT has grown, slowly a huge number of smart devices will be attached to edge computing(EC) and produce unlimited IoT traffic. Due to a vast volume of control messages, the controller may not have adequate capacity to respond to them. To handle such issues and to attain better load balancing, this work introduces a dynamic switch migration approach. This article has announced an efficient switch migration-based load balancing (ESMLB) framework, which targets allocating switches to an

underloaded controller efficiently. For selecting of target controller in this article technique for order preference by similarity to an ideal solution (TOPSIS) has been used. How to switch selected is not addressed in this article and performance assessment metrics such throughput, response time, and packet loss are not considered that affect the network performance.

The work in (Babbar et al., 2021) multiple controllers is used to solving the problems of the single controller such as being unable to handle heavy load traffic due to its restricted capacity. Multi-controllers on the control plane are used to realize quality network performance and robustness. One of the problems addressed in this paper is the constant configuration of the mapping between switch –controllers that result in unequal distribution of the load between controllers. To solve these issues software-defined vehicular networking (SDVN) has been involved as a configurable and scalable network and proposes a load balancing algorithm based on latency for multiple SDN controllers. By selecting the required latency and resolving multiple overloads concurrently, the challenges of controller load balancing are solved. The load balancing method is designed to measure the load of controllers and switches linked to the controller and inform the load information scientifically to the super controller through transferring messages which is the important part of the load balancer controller and to measure the controller load latency is used. But, the method in which switch and target controller are selected is not addressed in this article and is not efficient under high incoming traffic load.

The work in (Al-Tam & Correia, 2019), addresses the problem of static assignments between switch-controller in multiple controller SDN is discussed. Since the network load fluctuates both in time and space, the connection between switches and controllers should be adaptive to unexpected changes in the network. To solve this, dynamic switch-controller mapping is proposed in this article. Migrating switches from the high-loaded controller to light-loaded controllers brings adaptability and flexibility to the network but, simultaneously, deciding which migrated switches and target controllers, while keeping a balanced load in the network is still the problem in this article. In this work, a local search algorithm is offered that studies shift and swap moves and incorporates a controller solution shaking scheme. The result is shown only in terms of robustness, load balancing, and computation time. The performance of the network is not addressed in this work.

The work in (Xu et al., 2019), used BalCon and BalConPlus that shows dynamic allocation between switches and controllers can improve the efficiency of handling fluctuations in traffic

load. In particular, the author suggests BalCon and BalConPlus achieve controller load balancing between SDN controllers with low migration costs. BalCon is appropriate for situations where the network does not require serial processing of switch requests. The simulation shows that BalCon and BalConPlus decrease the load imbalance problems when only incoming traffic load is mice flow with little computing effort.

The work in (Hu, Lan, et al., 2019), suggests EASM balance controller loads and improves migration efficiency. The core of EASM is to migrate switches and balance controller loads considering performance, response time, load balancing rate, and migration efficiency. The outcomes depict that EASM at once achieves a low controller response time, high controller performance, low migration costs, and a better load balancing rate. But, not consider number packet loss and are also not efficient under high load incoming traffic loads.

The work in(C. Wang et al., 2017), raises improving migration efficiency by using SMDM. Based on the best migration efficiency situations. The SMDM method is designed by the greedy method, and the method consists of two phases which are load balancing detection and migration actions generation. A greedy-based method has been developed that offers the possible migration action options when a load imbalance occurs between the controllers. The load is calculated from the number of packet-in messages and the minimum path costs from the switch to the controller. The decision to migrate the switch is based on this. The output of this step provides a set of immigration controllers and emigration controllers.

The work in(Yeo et al., 2021), uses dynamic Switch migration for balancing load distribution between SDN controllers. This work is based on reinforcement learning, controller and switch selection system for switch migration, and known as SAR-LB is used. SAR-LB uses the utilization ratio of many resource types in both switches and controllers as the inputs of the neural network. Switches are also viewed as RL agents to decrease learning action while considering all cases of migrations. The result of the experiment shows that SAR-LB achieves an almost even load distribution between SDN controllers due to the precise decision-making during switch migration.

The work in (Li et al., 2019), discusses the challenges of multiple controller deployment on the control plane and divides the suitable subdomain for each controller. Each controller is accountable for centrally controlling switches in its domain. Multiple controllers work together with each other to attain efficient network management. However, multiple controllers setting

out cannot adapt to the dynamic changes of network traffic. If the switching traffic in the domain increases and decreases severely in a specific period, the load imbalance of the multiple controllers will be vast, resulting in low throughput, high response time, high packet loss, high latency, and other network performance degradation problems. To solve these issues this work proposes, an efficient dynamic load balancing scheme based on the Nash bargaining game model is proposed. First, connectivity of network node is considered, Then, the model sharply improves the two contradictory objectives of load balance and migration cost. Finally, the model is solved by an improved firefly algorithm and the optimal network mapping state is achieved. In this paper, the author does not consider the throughput and number of packet losses. How the controller is selected and the switch selected is not addressed in this work.

The work in (Hu, Yi, et al., 2019) proposes a switch migration-based decision making(SMDM) scheme that could solve the problem of load imbalance and metrics such as migration cost and load balancing rate are considered. To utilize the migration efficiency model an EASM method based on greedy technique is designed and therefore, guides the selection of suitable migration actions. SMDM framework dynamically balances the load distribution between the controllers and enhances migration efficiency. But, not considering the throughput and number packet rejected during migration degrades the system performance, and also the way switch and controller selected is not addressed.

The work in (Xue et al., 2019), addresses the problem of load balancing. The Ant-Colony Optimization (ACO) algorithm has been accepted to be effective for LB of SDN between numerous existing optimization algorithms. The merging searching optimal and latency results are the main criteria of ACO. In this work, a novel dynamic load balancing system that incorporates GA with ACO for additional improving the performance of SDN is used. The ACO improves the result in terms of the round trip time, rate of searching optimal path, and packet loss rate, but does not take into account the throughput and response time which affect the network performance in the SDN control plane.

The work in (Abdelaziz et al., 2017), proposed a novel clustered distributed controller architecture in the real setting of SDNs. The proposed comprises multiple common SDN controllers. The proposed method is assessed through a real-world network topology running on top of an emulated SDN environment. The result depicts that the proposed method in this article can meaningfully reduce the packet loss and the average latency compared to the

distributed controller without clustering. But, the dynamic mapping amongst switch and controller is not considered in this article, and also the performance assessment metrics such as throughput, response time, and packet loss are also not considered.

The work in (AL-Tam & Correia, 2019), addresses the problem of mapping the switch to controller relationship. Load balancing and stability are considered when establishing this mapping. Load balancing is important to enhance resources utilization, and mapping stability decreases the overhead in the control plane happened when exchanging information generated by new mappings. This article offerings a model for dynamic switch-controller mapping to realize high load balancing and reduce the number of new switch-controller assignments. Instead of migrating switches, the proposed method balances the load of the controllers by migrating fractions request flows among controllers, whereas reducing the number of new mappings. The model is expressed as convex quadratic programming, where the properties and the possibility of this model are examined mathematically. This work is not considered the other performance degradation metrics such as throughput, response time, and packet loss and also no mechanisms for switch and controller selections

The work in (Filali et al., 2019), studies the scheme of distributed architectures of the SDN control plane for balancing the load among controllers. Solutions planned for SDN load balancing commonly use the switch migration method. However, an efficient switch migration means generating the action at the right time and intelligently choosing the migrated switch and the target controller. This work proposes a switch migration scheduling algorithm to increase migration efficiency and confirm load balancing among controllers. Therefore, the author of this paper uses a multi-step ARIMA forecasting model to predict the long-term controller's load. When an overload is forecasted, a switch migration action is scheduled in improvement. After validating the accuracy of the ARIMA forecasting model, we assessed the performance of the algorithm by evaluating the response time of controllers. Static mapping among switch and controller is still challenging in this paper, and also other performance evaluation metrics such as throughput and packet loss are not considered in this works.

The work in (Y. Wang et al., 2017), addresses issues of load balancing in SDN. This article proposes a load balancing mechanism based on a Load Informing strategy (LILB). The mechanism involves load measurement, load informing, balance decision, and switch migration components. When the overloaded controller is happen, it can make decisions without collecting other controllers' load information. To decrease the communication excess and processing overhead initiated by the load informing component, this paper also recommends an inhibition algorithm to lower the frequency of informing load information. Furthermore, this paper designs approximately decision approaches of judging overloaded controllers, migrated switches, target controllers, and a judge almost accepting a migration request to avoid the load oscillation between controllers. In the meantime, to attain the smooth switching of controllers' roles during migrating switches, an information interaction technique is also designed. This work is only working under a small-scale network that is not efficient when traffic flow is high. So this, need to be taken into consideration in the future.

Generally, SSMS is associated with the problem of increase in congestion when chosen the target controller while the CAMD method of switch selection is associated with the problem of switch selection inefficiency, low throughput, high response time, and high packet loss. This work aims to address the abovementioned difficulties associated with the CAMD and SSMS methods. The proposed ISMM is expected to select a heavily-loaded switch for migration from the overloaded controller but will migrate it to the most appropriate controller during migration so that that maximum clustered resources would be left free.

Table 2.3 Summary of Related Works

Ref and year	Method	Contribution	Limitation
(Yang Zhou et al., 2018)	ElastiCon	ElastiCon can cover purposefully important controllers by distributing the difference of traffic load between controllers and designed for Small scale networks. • Floodlight is the controller used.	-The method was not set methods to select an appropriate switch and the target controller for migration. -Response time is considered
(Xu et al., 2019)	BalCon and BalConPlus	This method, shows the dynamic between switches and controllers offer system elasticity and efficiency under varied traffic loads. • RYU controller.	-Appropriate only for small-scale networks. -Throughput and response time of the controller are not considered to balance the load.
(Sahoo & Sahoo, 2019)	CAMD	This method is effective and receives more loads when the incoming traffic is small flows. • Floodlight controller.	-Static map between switch and controller. -Inefficient when incoming traffic is between 501-5010pps.
(Hu et al., 2017)	SSMS	This method decides overloaded and underloaded controllers based on the greedy method and migrates switch sequentially. • OpenDayLight controller.	-Controller response time, throughput, and packet loss are not evaluated. -inefficient when incoming traffic is 501-5010pps.
(Hu, Lan, et al., 2019)	EASM	EASM balances the controller loads and improves migration efficiency by migrating switches considering LBR. • OpenDaylight controller.	-The number of packet loss is not considered. -Inefficient when incoming load traffic between 501pps-5010pps.
(C. Wang et al., 2017)	SMDM	In this method, the migration plan is expressed as a set of migration actions and the greedy method was designed to apply the migration efficiency model. • OpenDaylight is the controller used.	-Controller throughput and packet loss are not measured that affects network performance -Under large-scale incoming load traffic it's not efficient.

2.7. Summary

In summary, many SDN-based multiple controller load balancing approaches have been suggested in recent years. The solution raised above is still suffering from certain shortcomings. To solve the challenges of controller load imbalance for multiple controllers in SDN two methods, i.e. controller clustering and switch migration are used. The switch migration methods were proposed for solving addressed problems in this thesis, because of their efficiency over controller clustering methods. Switch migration solves the challenges based on modules and considers different performance assessment metrics to increase the network performance. Different researchers consider different evaluations of performance metrics for their works. But, in this thesis, throughput, response time, and packet loss were considered. Some existing work considers these metrics but not effective under high incoming traffic loads, which means that it needs to improve to solve problems efficiently.

Therefore, based on the above-stated gaps of the related works, the problems are solved by using an ISMM. Moreover, it focuses on improving the method for performance enhancement and overcomes the gaps of some researches. Also, put how switch and controller are selected for performing the migration. In the next section of chapter three, the proposed solution was discussed.

CHAPTER THREE

3. PROPOSED IMPROVED SWITCH MIGRATION METHOD

3.1. Introduction

In this chapter, the proposed solution for the controller load imbalance problem between multiple controllers includes an overview of the proposed solution, general design of the proposed solution, flowchart of the proposed method, dynamic controller threshold, and the general assumption is discussed in detail. Furthermore, switch selection, target controller selection, and dynamic threshold adjustment for the controller are also discussed.

3.2. Proposed Solution

3.2.1. Overview of Improved Switch Migration Method

Switch migration is a way of handing over the functionalities of the switch that is handled by one controller to another controller. This migrating of switches from the loaded controller to underloaded controllers are used for balancing load between the controllers. It supports the dynamic adjustment of the controller load. The existing switch migration designs are difficult to realize good load balancing performance. Several research studies show that traffic conditions are different both in terms of time and space. This makes some controllers overload while others are under-loaded. If the load on a controller exceeds its maximum controller threshold, the switches on the overloaded controller are migrated to the underloaded controller (H. Wang et al., 2018).

In this study, OpenFlow v1.3 was used for communication between switches and controllers. The OpenFlow 1.3 protocol offers the way of applying a switch migration and defines three roles master, slave, and equal for the controller. A switch may be connected to multiple controllers in the domain as a slave or equal role and one controller as a master role. This is why we have proposed a solution that reduces the controller loads by switch migration in our study (Hu, Lan, et al., 2019)(Beiruti & Ganjali, 2019).

Figure 3.1 shows the overview of the switch migration process between two controllers, namely, controller 1 and controller 2. In this switch migration process, Switch 3 is migrated from controller 1 to controller 2 to balance the load between them.

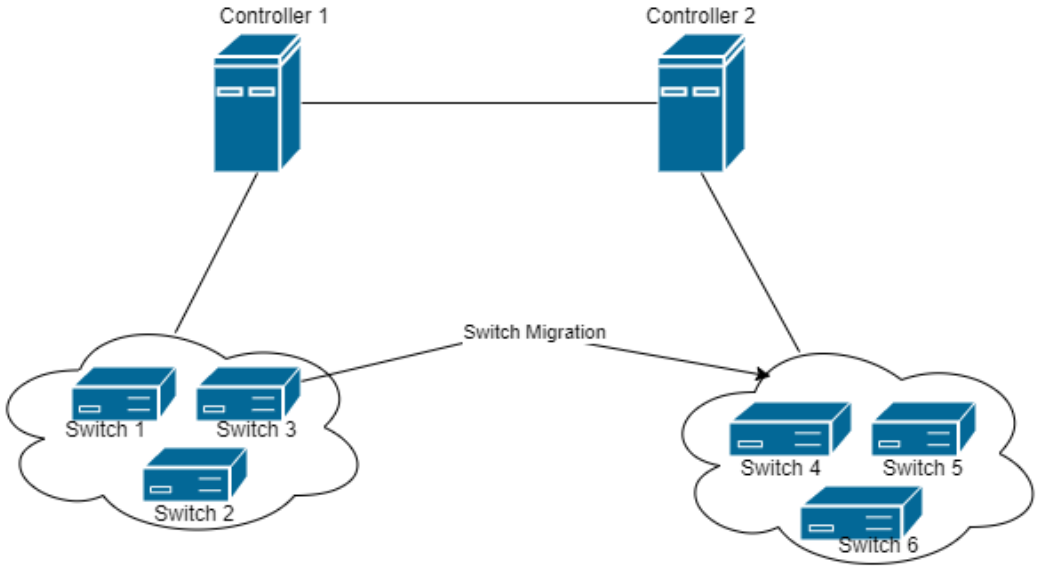


Figure 3.1 Overview of Switch Migration Method

One of the efficient ways of solving problems of load imbalance for multiple controllers is through the switch migration method. The proposed solution then balances load distribution among the controllers to achieve efficient load balancing and improve the network performance. If not it will degrade the system performance. When the controller load exceeds the set of defined controller thresholds, the controller will migrate the switch to the controller with a minimum load. First, the load is measured and judgment made, second, the switch is selected, third a target controller must be specified to send the migration request, then the request is checked by the controller and if possible it's accepted. Then, carry out switch migrations from the overloaded controller to the underloaded controller. Generally, this solution increase controller throughput improves controller performance, reduces controller response time, and reduces packet loss after load-balanced. The overall design of the proposed solution is explained in detail in Section 3.3.2 with its four modules.

3.2.2. Overall Design of Proposed Solution

In the multiple controller SDN network, there is static traffic that varies with spatial and temporal features that are prone to controller load imbalance. There is also unbalanced load distribution between multiple controllers. To resolve those challenges, the existing switch migration method was improved, and proposed an improved switch migration method that balances load efficiently. The overall design of the proposed solution is shown in figure 3.2, which consists of four modules that run locally on each controller in the network topology to balance loads.

The modules in the design include the load measurement and judgment module, the switch selection module, the controller selection module, and the switch migration module.

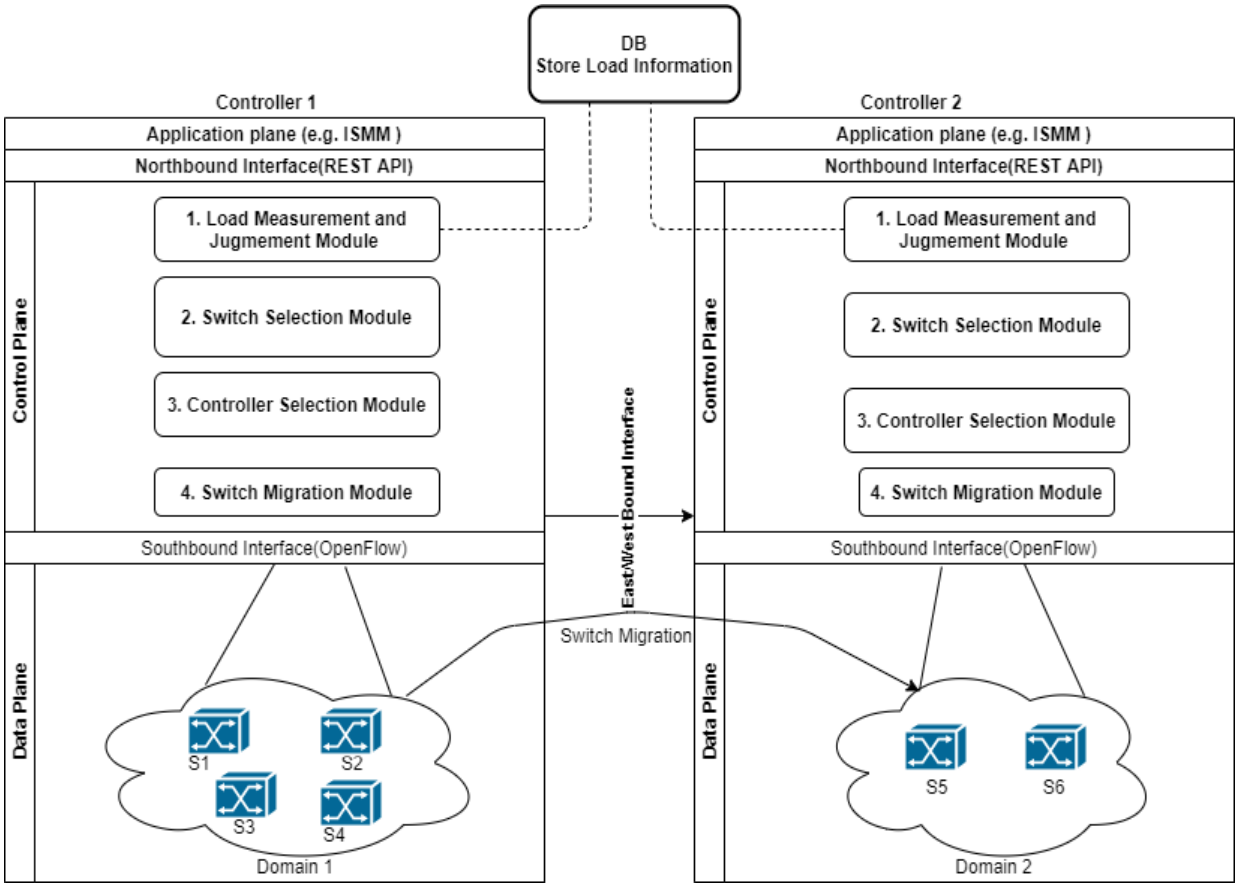


Figure 3.2 Overall Architecture of Proposed Solution

Figure 3.2 shows, the overall design of ISMM which contains four modules that run locally for each controller in the network topology. The load measurement and judgment module in this design check immediately whether the controller load surpasses the predefined threshold and make decisions. Each controller in the network design collects all controller loads and calculates the controller's load mean which is used to decide the set of underloaded controllers in the network topology. Based on this the controllers which load capacity is less than or equal to controllers load is defined as a set of underloaded controllers.

Accordingly, the switch selection module is also becomes activated next to the load measurement and judgment module and its main work is to select the highest loaded switch from the set of overloaded controllers for the migration procedure which improves the selection efficiency. So, that the load reduction of the overloaded controller is not greater than the load increase of the target controller to have a successfully balanced controller load. The third

module, the controller selection module also gets started beside modules 1 and 2 to ensure the most suitable controller from a set of underloaded controllers. Finally, after measuring load and making a judgment, switch selected, controller selected both underload and overload, the fourth module is started to accomplish switch migration and balance loads.

Controller threshold and load statistics on multiple controllers stored for easy access and east-west bound interface are managing the services that stick the communication between the multiple controllers as shown in the figure above. In the designed figure 3.2 controller1 under domain 1 will migrate its switch to controller 2 under domain 2 to balance the load. When the current load status of the controllers surpasses the specified threshold (T), the method immediately calls module 1 to calculate the load status of all controllers in the network topology and make a judgment. Then, simultaneously calls module 2 for switch selection and module 3 for controller selection. Then, the method takes a decision, and module 4 performs actions by migrating the selected switch to the target controller.

The generalized pseudocode of the proposed ISMM was described in **Pseudocode 3.1.**

Pseudocode 3.1: Pseudocode of the Proposed ISMM

Step 1: **Input:** C, T, CL_{cur}

Step 2: **Output:** Balanced Load Between Controllers

Step 3: **if** (CL_{cur} > T) **then**

Step 4: Initialize Load Measurement and Judgment Module

Step 5: Initialize Switch Selection Module

Step 6: Initialize Target Controller Selection Module

Step 7: Send Role Request to Controller

Step 8: **if** (Role Request Accepted) **then**

Step 9: Switch Migration Module do ($C_j \leftarrow S_k$)

Step 10: Update C Load (both current and target controller)

Step 11: **end if**

Step 12: **end if**

Step 13: **Display** Load Balancing Success

Where **C:** a set of controller **T:** Controller load threshold **CL_{cur}:** Current controller load

3.2.3. Flow Chart of Proposed ISMM

Proposed ISMM calculates a load of each controller regularly, which is calculated from the incoming traffic loads. When the controller load passes the threshold, the switch with the highest load and target controller is selected and turned on the migration, the request will be sent to the target controller from the overloaded controller. If the migration request will be accepted by the target controller and the switch will be migrated to the target controller and the role of the controller is changed from master to slave and vice versa. The flowchart of the proposed ISMM is revealed in figure 3.3.

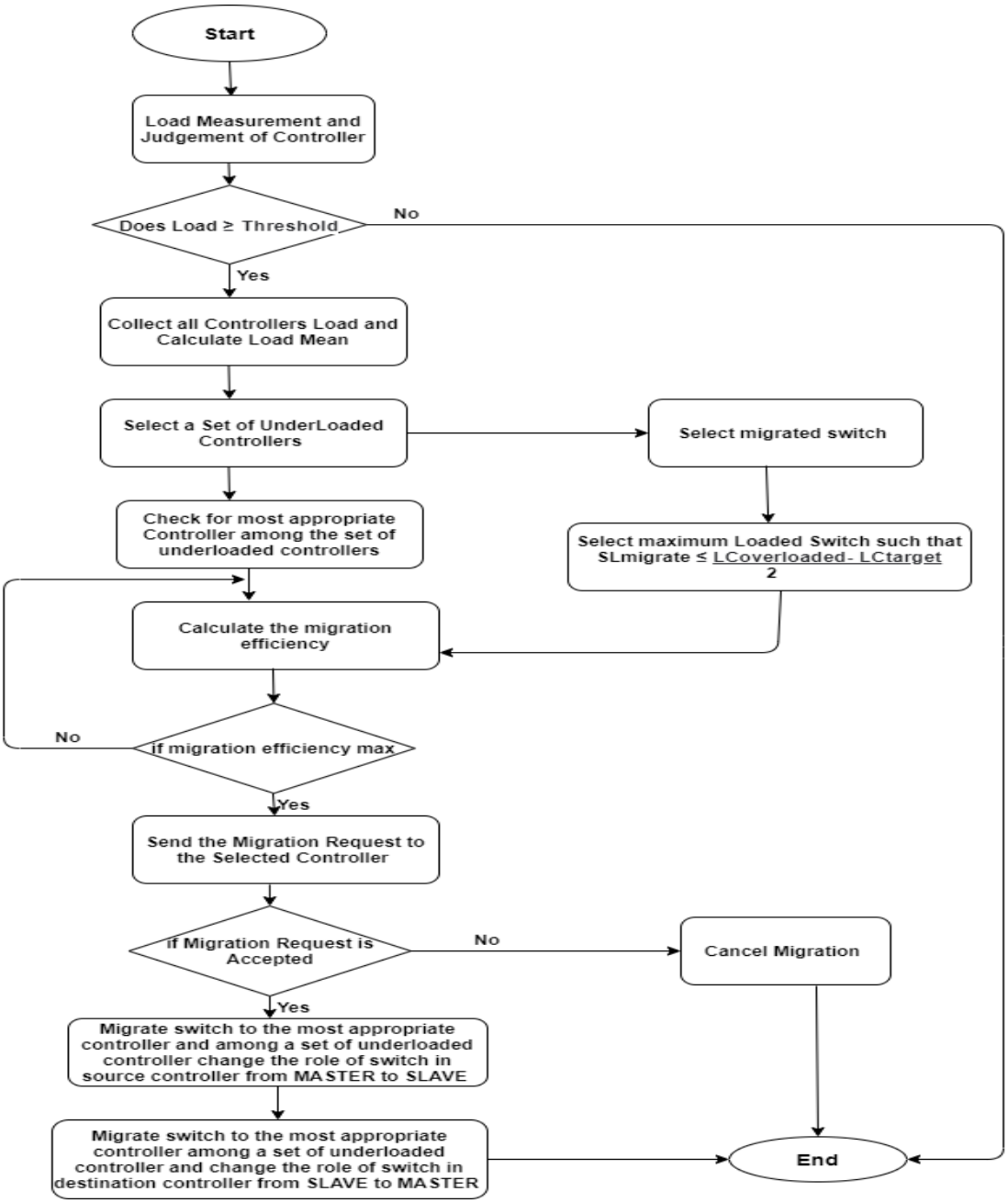


Figure 3.3 The Flow Chart of Proposed ISMM

3.3. Flow Description of ISMM

Step 1: Load Measurement and Judgment Module

This module tracks and reports the load statistics of each controller in real-time. The load statistics include controller resources such as memory usage, CPU usage, and bandwidth usage. If the controller load surpasses a predefined threshold in **Pseudocode 3.2**, it estimates the load of all other controllers and then identifies the set of the under-loaded controller using system load variance which indicates the system performance error. The smaller the system performance error indicates the better the system performance.

As described by the author in (LI et al., 2018), there are two methods in getting the load status of SDN controllers. These methods are the centralized load collection method and the distributed load collection method. In a centralized load collection method, the load balancer is deployed to track and gather the load status of the controller. This method is not efficient enough and can only apply to a smaller network environment. However, every controller node in the distributed load collection method can act as a load balancer. It can also easily track, estimate, and report its load information regardless of the size of the network which makes it more appropriate and applicable in a large-size network.

Pseudocode 3.2: Pseudocode for Load Measurement and Judgment Module of ISMM

Step 1**: Input:** n, T

Step 2: **Output:** C_{UL} and C_{OL}

Step 3: max= 0

Step 4: **for** i=1 to n **do**

Step 5: Read Controller Load CL ($_{ci}$)

Step 6: max = max + CL ($_{ci}$)

Step 7: **if** CL ($_{ci}$) > T **then**

Step 8: **end if**

Step 9: **end for**

Step 10: Find Mean Load $\overline{\mathrm{CL}}(\mathrm{Ci}) = \frac{1}{n}\sum_{i=1}^{n} CL(ci)$

Step 11: find load variance $\lambda = \frac{1}{n}\sum_{i=1}^{n}\left(\mathrm{CL}(\mathrm{Ci}) - \overline{\mathrm{CL}}(\mathrm{Ci})\right)^2$

Step 12: **if** CL ($_{ci}$) < T

Step 13: $C_{UL} \leftarrow C_i$

Step 14: **if** CL ($_{ci}$) > T

Step 15: $C_{OL} \leftarrow C_i$

Step 16: **end if**

Step 17: **end if**

Step 18: **Display** C_{UL} and C_{OL}

Where, **n:** number of controllers, **T:** Threshold, **C_{OL}:** Over-Loaded Controller **C_{UL}:** Under-Loaded Controller, **$CL_{(Ci)}$:** Load of Controller C_i

The load on the controller can be measured in two ways. These two ways are performance metrics and switch input metrics. Metrics like message flow rate, response time, and the flow table entries are commonly used by switch input metrics in telling the load condition of the controller. However, this study does not use switch input metrics since it will directly affect the controller resources. The performance metrics related to the controller resources such as memory usage, CPU usage, and bandwidth usage are considered in this work to describe the load status of a controller. The packet-in message from switches to the controller is taken as the input load from a switch. All controller in the domain guesses their total load of $CL_{(ci)}$ based on the packet-in message. This can be achieved in real-time based on the metrics stated above with their equivalent assigned weights which can be defined in equation 3.1.

$$CL_{(ci)} = [w1 \quad w2 \quad w3] \begin{bmatrix} CLband(Ci) \\ CLmem(Ci \\ CLcpu(Ci) \end{bmatrix} \qquad \textbf{\textit{Equation 3.1}}$$

Where, w1, w2, and w3 are the weights memory, bandwidth, and CPU of control plane resources, and $CL_{band}(Ci)$, $CL_{mem}(Ci)$, and $CL_{cpu}(Ci)$ represent the actual bandwidth, memory, and CPU utilized on the control plane by switch packet-in request.

Supposing that, θ_k is the packet-in message created by switch S_k to controller C_i during the period, t. Weight is allocated to controller resources now to merge the similarities and differences between resources that are used and to know the true consumption of resources by the switch. The main significance of weight to controller resources is to minimize bias. Without the weight, the inconsistency will grow rapidly and increase general system error. Therefore, the total load on the controller calculated using the packet-in messages can be expressed in equation 3.2.

$$CL(ci) = \sum_{Sk \in Ci} \theta k(t) \alpha ki \qquad \textbf{\textit{Equation 3.2}}$$

Where**, CL $_{(Ci)}$:** the current load status of controller C_i, **αki**: control plane resource utilization value, and **θk (t):** the packet-in messages sent by the switch S_k to controller C_i at time frame t. The value of αki is expressed as the weighted sum of network utilization generated by each switch S_k to controller C_i and can be described as equation 3.3.

$$\alpha ki = \omega i^{x} \frac{\Theta k * Xi}{\mu i} + \omega i^{y} \frac{\Theta k * Yi}{\nu i} + \omega i^{z} \frac{\Theta k * Zi}{\tau i} \qquad \textbf{\textit{Equation 3.3}}$$

Where $\boldsymbol{\theta_k}$ is the packet-in message produced by the connected switches. $\mathbf{X_i}$**:** CPU usage, $\mathbf{Y_i}$**:** memory usage, and $\mathbf{Z_i}$**:** bandwidth usage of C_i per packet-in for switch S_k. $\boldsymbol{\omega_i^x}$**,** $\boldsymbol{\omega_i^y}$, and $\boldsymbol{\omega_i^z}$ denote the predictable weights of controller resources. The CPU usage, memory usage, and bandwidth usage must satisfy the situation $\boldsymbol{\omega_i^x + \omega_i^y, + \omega_i^z = 1}$. Also, μi, νi, and τi denote the capacity of the controller (C_i) that is CPU, memory, and bandwidth respectively.

The variance is announced to approximation the controller equilibrium level since all controllers are measured is calculated by equation 3.4.

$$\lambda = \frac{1}{n} \sum_{i=1}^{n} \left(CL(Ci) - \overline{CL}(Ci)\right)^2 \qquad \textbf{\textit{Equation 3.4}}$$

Where, **:** Variance, **CL (C_i):** controller load status on real-time, $\boldsymbol{\overline{CL}(Ci)}$ controller mean load.

Based on this variance calculation the migration efficiency is identified. Migration efficiency calculated is used for knowing the most appropriate controller for the set of underloaded controllers. If the migration efficiency is maximum, that controller with maximum migration efficiency is selected as the target controller.

Step 2: Switch Selection Module

Maximum load switch is selected from overloaded controllers for migration purposes in this module. Selecting the highest loaded packet in the request switch will bring in important enhancement in network efficiency. This is justified using **equation 3.6** in our study. According to the OpenFlow protocol v1.3 used in this work, the roles between controllers are changed from master to slave and vice versa. The controller load is calculated by the amount of packet-in messages that are directed by the connected switches in a time interval, t. various messages like packet-in messages, hello messages, echo messages, and others can be forwarded from the switch to the controller. However, to represent the load of the controller, packet-in messages send from switch to controller are used in our work.

This study supposed that the load reduction of the overloaded controller, should not be more than the increase of the target controller load to have a successful get-balanced system. Subsequently, this study summarized that selection of switch is made under the condition of equation 3.5:

$$SLmigrate \leq \frac{CLoverloaded - CLtarget}{2} \qquad \textit{Equation 3.5}$$

Where, $\mathbf{SL_{migrate}}$**:** the load of the migrated switch from the overloaded controller, $\mathbf{CL_{overloaded}}$**:** the load of the overloaded controller, and $\mathbf{CL_{target}}$: the load of the most appropriate controller which is the target controller.

Pseudocode 3.3: Pseudocode for Switch Selection module of ISMM

Step 1: **Input:** C_{OL}

Step 2: **Output:** Migrated Switches S_k from C_{OL}

Step 3: List switch in C_{OL} with their incoming packet-in load

Step 4: Initialize switch set P= {}

Step 5: **for** ∀ $S_k \in C_{OL}$ **do**

Step 6: **for** ∀ packet-in flows $S_k \in C_{OL}$ **do**

Step 7: Find migrated Switch from List of C_{OL}

Step 8: **end for**

Step 9: **end for**

Step 10: Select Maximum Loaded Controller from C_{OL} based on selection efficiency in equation 3.6.

Step 11: S_K = max $\{P_{sk}\}$, $S_k \in C_{OL}$

Step 12: Migrate $\mathbf{SLmigrate} \leq \frac{\mathbf{CLoverloaded - CLtarger}}{\mathbf{2}}$

Step 13: **Display** switch set P={}

Where, $\mathbf{C_{OL}}$**:** Overloaded Controller, Sk**:** migrated switch

This work shows how the LBR is affected by choosing a switch with fewer flow requests. This work had considered equation 3.6 for the evaluation of LBR which is defined as

$$\text{LBR} = \left(\frac{1}{n}\sum_{i=1}^{n}(Ci)\right) / \max_{i=1} n\,(Ci) \qquad \textit{Equation 3.6}$$

Where, **LBR:** Load Balancing Rate, n**:** Number of Controllers, $\mathbf{C_i}$**:** list of controllers load, max: Maximum loaded switches in C_{OL}

From the above equation, this study shows that selecting a switch with the highest flow request will bring the most significant improvement in network efficiency.

A switch with the maximum loaded flow requests from an overloaded controller is considered in this study during the migration module, as this will LBR of the network. LBR is the degree of closeness to the real load distribution and can be measured with a load balancing rate. This equation indicates that the maximum loaded switch brings better improvement in the network efficiency. So, based on this justification we select maximum loaded switches for migration.

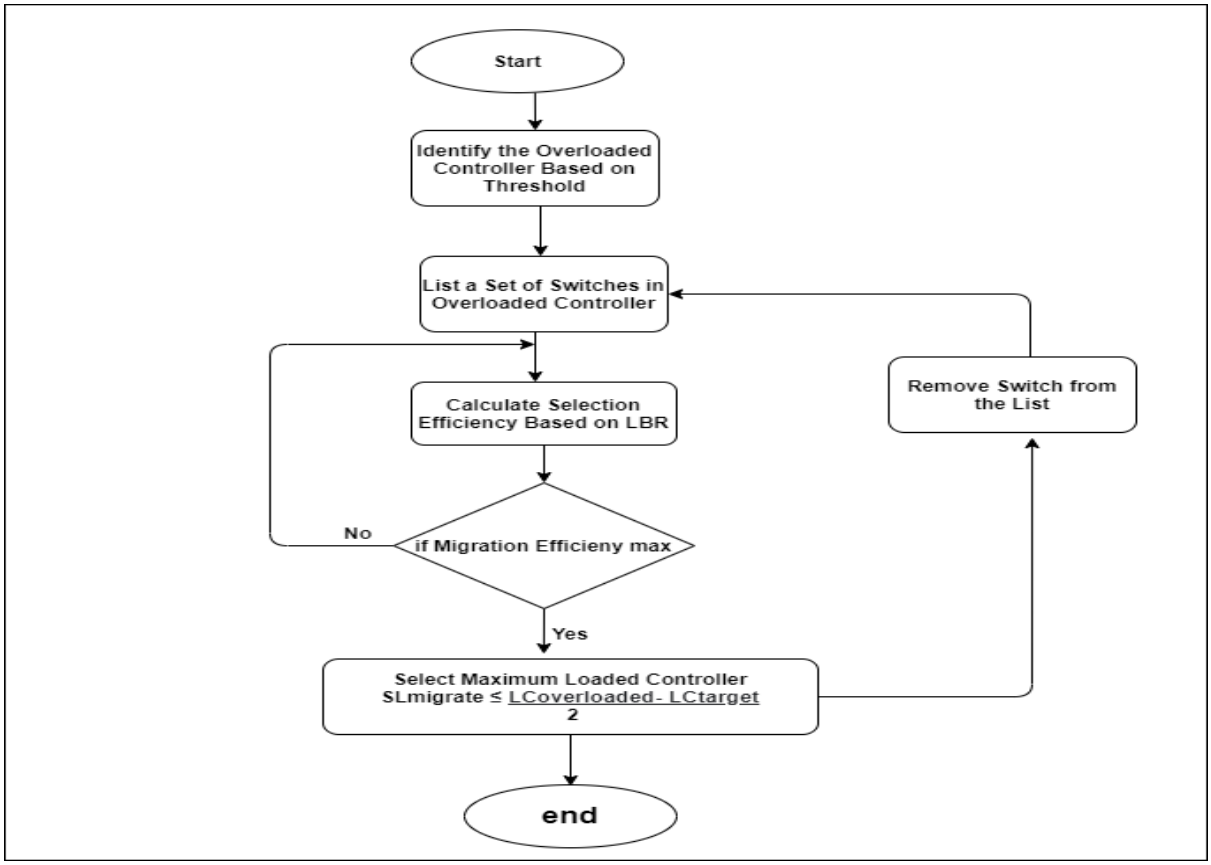


Figure 3.4 Flowchart of Switch Selection

Step 3: Controller Selection Module

After selecting a switch to be migrated, the next step is to find the target controller. We have been considered the incoming packet-in requests to the controller to find the best selection. The load measurement and judgment check the load on the controller and calculate each controller load. If the weight of the controller exceeds the threshold, then the overloaded controller is known. If the weight does not surpass the threshold, the underloaded controller in the control plane will be selected as the target controller for switch migration depicted in the flow chart of figure 3.5.

Pseudocode 3.4: Pseudocode for Target Controller Selection Module of ISMM

Step 1: **Input:** S_K, C_{UL}

Step 2: **Output:** Target Controller, C_j

Step 3: Get the Current Load on C_{UL}

Step 4: **for** Controller $C_j \in C_{UL}$ **do**

Step 5: **for** j=1 to n $\in C_{UL}$ **do**

Step 6: Calculate Load Variance using $\nu = \frac{1}{n}\sum_{i=1}^{n}\left(\mathrm{CL(Ci)} - \overline{\mathrm{CL}}\mathrm{(Ci)}\right)^2$

Step 7: **end for**

Step 8: **end for**

Step 9: Check that $CL_{(Ci)} + SL_{(Si)} \leq C_L\ (c_j)$

Step 10: $C_{TS} = \max\{C_{UL}\}: (C_i) + SL_{(Si)} < CL_{(Cj)}$, **where** $C_j \in C_{UL}$

Step 11: **Update** C_{UL} with current Load Status

Step 12: **Display** Target Controller, C_j

Where **C_{OL}:** Overloaded Controller **C_{UL}:** Underloaded Controller **S_k:** migrated switches **C_j:** Target controller

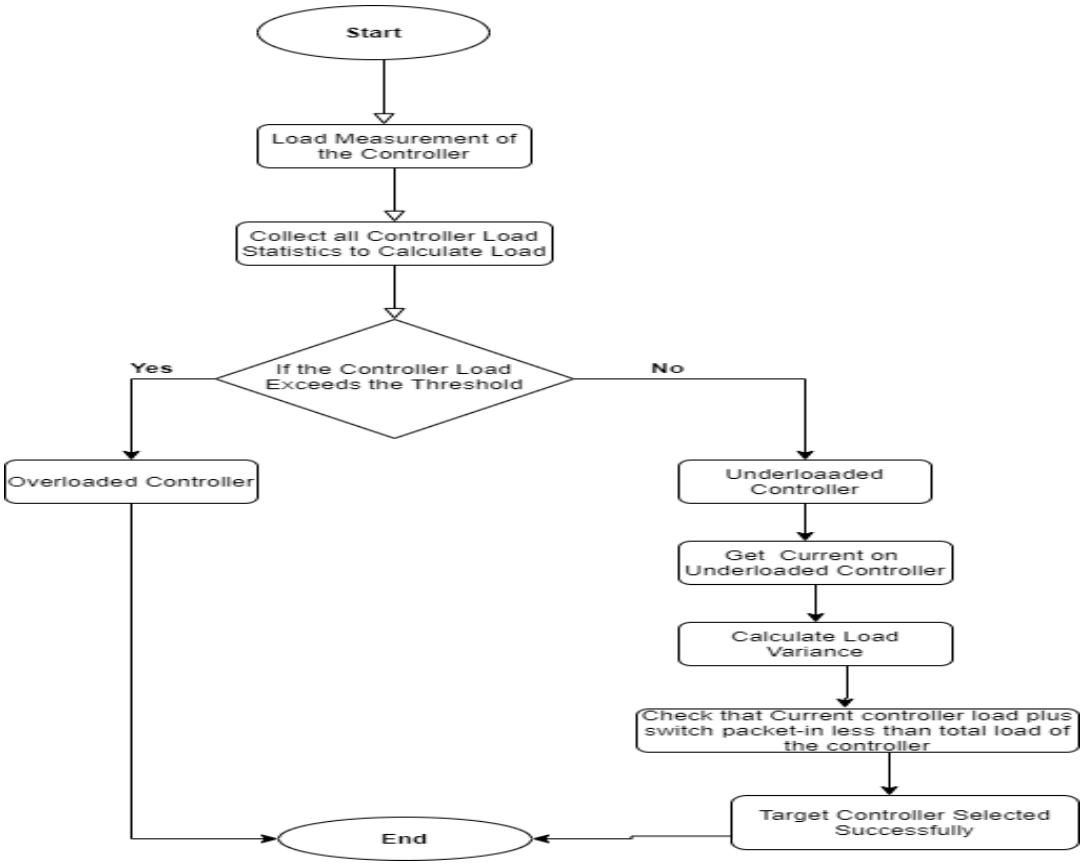


Figure 3.5 Flowchart of Controller Selection

The overloaded controller is the controller which switch is migrated from that acts as an emigration controller or initial controller to quickly balance the controller loads. Therefore we calculate the load between all controllers. After calculating load, we assume L (Cm)>L(Cn) or L(Cm)>T, and then set Cm as the emigration controller in this study.

Step 4: Switch Migration Module.

It is the core part of the ISMM design, which is used for managing and finalizing the switch migration process.

3.4. General Assumption

The following general assumptions are made to be considered in the ISMM while solving the problems of controller load imbalance.

Assumption 1: The controllers can talk with each other to share the load and each switch in the network can access each of the controllers to migrate the switch from one to another.

Assumption 2: Once a switch has chosen to migrate, it cannot revert to the previous controller's domain until the new controller's domain has met all of its control requirements.

Assumption 3: Only one controller is assigned to each switch as master controller and the rest runs in equal or slave mode.

Assumption 4: All the controllers cannot be overloaded at the same period, t.

Assumption 5: At a time, t, only one switch can migrate.

Assumption 6: To simplify our experiment, all controllers have the same capacity.

3.5. Dynamic Controller Threshold

A threshold value is required to identify the overloaded controller and the happening of load imbalances. Individually controller in the network checks whether its load exceeds a predefined threshold set by the user. If the situation is true; guess the average load of all connected controllers in the network. If the estimated average value is less than or equal to the predefined initial threshold, the initial threshold is returned. Then, set the estimated average load as a new threshold and update it dynamically. ISMM needs to collect every other controller load in the system to make appropriate load balancing decisions. However, a static controller threshold method can reduce system performance. To solve the problem of reduced system performance, this study had proposed a dynamic load collection threshold method presented in **Pseudocode 3.5.** After that, dynamically update the controller threshold.

Pseudocode ***3.5: Dynamic Controller Threshold using ISMM***

Step 1: ***Input:*** $CL_1, CL_2, \ldots, CL_n$

Step 2: ***Output:*** Adjustable T

Step 3: Find Mean Load $\bar{C}L\ (Ci) = \frac{1}{n}\sum_{i=1}^{n} CL(Ci)$

Step 4: Let initial T = 1800

Step 5: **If** $\bar{C}L\ (Ci) \leq$ initial T **then**

Step 6: T=1800

Step 7: **If** $CL\ (Ci) \geq$ Initial T **then**

Step 8: $T= \bar{C}L\ (Ci)$

Step 9: **end if**

Step 10: **end if**

Step 11: **Display** T

Step 12: end procedure

Where **T:** Threshold **$CL_1, CL_2 \ldots, CL_n$:** List of controller load **n:** number of controllers

This study presented a dynamic controller threshold that updates of controller threshold in the network which supports dynamic connection among switch and controller and improves the network performance.

3.6. Summary

The proposed ISMM migrates the switches from one controller to another controller's domain to reduce its load and improve the performance of the network. It contains four different modules including a load measurement and judgment module, a switch selection module, a Target controller selection module, and a switch migration module. The migration is done when the load on the controller exceeds the controller threshold. Based on the four-module in ISMM the problem of controller load imbalance is solved and improves the overall system performance. The controller threshold is also updated dynamically to solve the problem of static mapping between switch and controllers. The proposed solution in this work has added a modification to the original code of existing switch migration and this was done in the Mininet.

CHAPTER FOUR

4. SIMULATION AND PERFORMANCE EVALUATION

4.1. Introduction

In this chapter, the environment used to conduct a simulation to solve the problem of controller load imbalance, simulation tool used, SDN controllers used, OpenFlow Protocol used, performance evaluation metrics, simulation results, evaluation of results, and discussion will be made in detail. Lastly, the performance of the proposed solution with two other methods discussed in related work is compared. The performance evaluation metrics such as throughput, response time, and packet loss are considered.

4.2. Simulation Tools in SDN

Various tools are available for simulation in an SDN network environment for solving load imbalance problems. The main SDN simulation tools include Matlab and Simulink, POCO, Mininet, Network Simulator version 3(NS3), OFNet (OpenFlow Network), and EstiNet11 network simulator are discussed.

4.2.1. MATLAB and Simulink

Matlab Simulink is the MATLAB tool that does not support OpenFlow and can be extended to support SDN. The OpenFlow protocol used for communication between two SDN planes is not supported by Matlab Simulink, so this tool is not used for controller load balancing(*Simulink Documentation*, n.d.).

4.2.2. Pareto Optimal Controller Placement (POCO)

It is most commonly used for controller placement problems that provide a convenient and easy way to calculate, analyze and visualize controller placement. It is used for the evaluation of the given topology. In this research study, we don't use these simulation tools (Hock et al., 2013).

4.2.3. NS-3 Network Simulator

NS-3 is designed mainly for educational and research use. It is a difficult tool that runs simulations defined by code generated by users, so we don't use these simulation tools in our research, because not fully support the OpenFlow protocol (*Ns-3 | a Discrete-Event Network Simulator for Internet Systems*, n.d.).

4.2.4. OpenFlow Network (OFNet)

OFNet is a toolbox that offers an environment for modeling OpenFlow networks like Mininet. OFNet is used to monitor OpenFlow messages, visual debugging of controller transactions, and traffic generation on an emulated network and assess the performance of the SDN controller. Even if its GUI is attractive, it supports less when compared with Mininet network simulation tools while controller load balancing.

4.3.5. EstiNet11 Network Emulator

ESTiNet11 is a commercial simulation and emulation tool, which is the best OpenFlow in the world for doing SDN research but it doesn't support controller load imbalance problems (*EstiNet - Simulator | EstiNet*, n.d.).

4.3.6. Mininet Simulation Tool

Mininet is established to support education and research in SDN. Mininet is a popular open-source network simulator capable of creating networks virtual hosts, controllers, switches, prototyping, testing, and SDN controllers. It was created in python programming language and provides a python API for user customization and can run in a Linux version Ubuntu 20.04.3 LTS. Our experiments are conducted on Mininet 2.3.0 and ODL was used which is an open-source SDN controller implemented in java (*Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet*, n.d.). Appendixes C shows the Mininet startup command and its interfaces.

Generally, from the aforementioned simulation tools, Mininet Simulation tools are selected for our work because Mininet outperforms other simulators in terms of:

- Computational Cost and Time.
- Memory Usage
- OpenFlow Support
- Popularity to solve the problem.
- Support SDN implemented architecture
- Open source tools

As described above some reasons, we worked our simulation on Mininet Simulator that was the best simulation tool and recommended for SDN researches. Because of its full support of OpenFlow protocols with their roles. The other simulation tools have lack full support of

OpenFlow protocol an also their computational time and cost, and memory usage is high. These advantages led us to the use of Mininet in our Study.

4.3. SDN Controllers

An SDN controller is a plane in SDN design that controls requests to improve network management and application performance. The SDN controller uses protocols to express switch where to send packets. Controllers in SDN send load according to forwarding rules set by a network operator, which minimizes manual configurations for single network devices. Any communication between the application and the data plane must go through the controller(Zhu et al., 2019). The controller exchanges data with applications such as load balancers or firewalls through northbound interfaces. The network devices communicate with controller communicates with controllers using an SBI, i.e. OpenFlow protocol (Mostafavi & Hakami, 2019).

There are two types of SDN controller's i.e. single controllers and multiple controllers. In our thesis, multiple controllers that are centralized logical views of the entire network are deeply studied. Today, there exist many SDN controllers which include OpenDayLight (ODL), Open Network Operating System (ONOS), RYU, Beacon, Floodlight, NOX, POX, Maestro, Kandoo, and others (Abdullah et al., 2018).

From the list of these controllers, ODL and ONOS support multiple controllers in SDN. The left controller supports a single centralized controller. So, compare ODL and ONOS, then select the most appropriate controllers to solve our implementation and used in our work is selected. Among the available controllers, there are some reasons to choose ODL:

- It supports SDN that automates and customizes networks of any size and scale.
- It builds robust open-source code that meets most SDN and NFV requirements.
- The industrial support of this platform encompasses the main part of the most powerful companies in the IT field like Cisco, Dell, HP, IBM, Intel, Juniper, Microsoft, Redhat, NEC, VMware, and Oracle.
- Provide Novel functionalities as a new platform, and implement some innovative functionalities concerning other controllers.
- Support multiple controller architecture.
- Modularity

4.4. OpenFlow Protocol

OpenFlow is a southbound API protocol that permits communication among control and data plane via the OpenFlow channel. OpenFlow protocols have been managed since 2011 by ONF, an organization dedicated to introducing and promoting SDN and NFV. There is a different version of OpenFlow, but a multi-controller architecture is only defined beginning from v1.2 of the OpenFlow protocol to improve communication among controllers and network devices. In figure 4.1, OpenFlow switch is software or hardware that forwards packets based on the controller in an SDN environment. In an OpenFlow switch, there are one or more flow tables, and each flow table has many flow entries. A flow entry in a flow table is used to match and process packets. It uses an OpenFlow channel to secure messages between the switch and the controller.

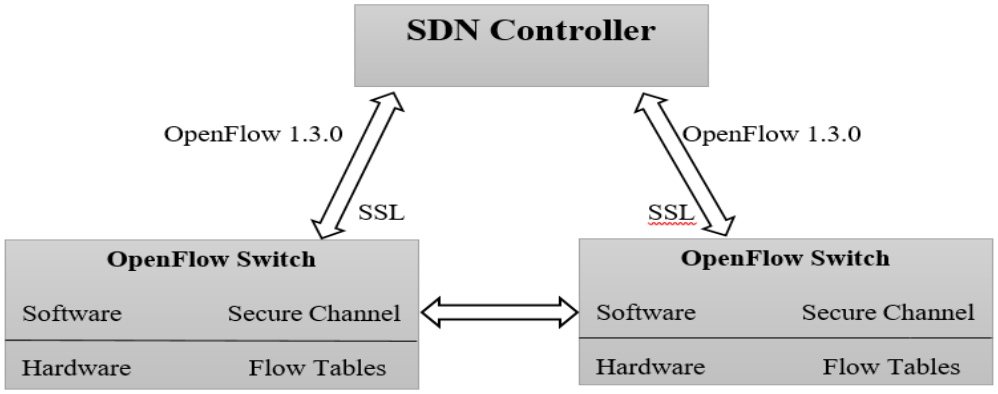


Figure 4.1 Communication between SDN controllers and OpenFlow Switch architecture

OpenFlow v1.3, used in this study because:

- Support different roles of controllers and switch migration. One OpenFlow switch can connect to more than one controller, and each of them could be in three different roles which are Master, Equal, and Slave. The controller in slave role contact to the switch is read-only, and cannot receive any packet-in messages. The controller in master modes can change the role of the switch and receive packet-in messages from switches.
- More flexible

As revealed in figure 4.2, switch S connects to two controllers A, B. Initially controller A is master and controller B is slave, after migration, controller B becomes master, and controller A becomes a slave. In this method, the switch S has been migrated from A to B.

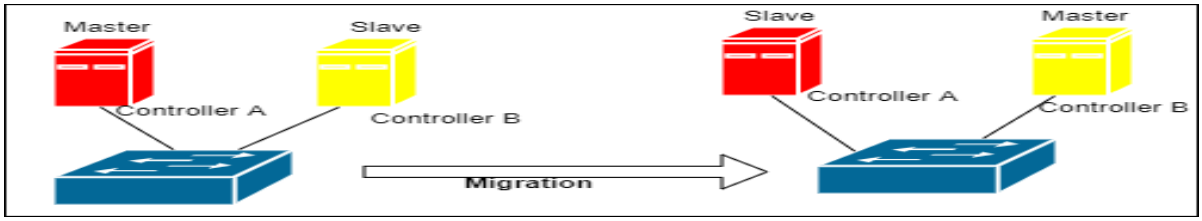


Figure 4.2 OpenFlow controller role Changing Process

Due to the significant benefits of OFv1.3 used for communication among controllers and switches in our work (Blial et al., 2016).

4.5. Performance Evaluation Metrics and Simulation Setup

4.5.1. Performance Evaluation Metrics

The performance assessment metrics were used to compare the simulation results. These performance assessment metrics were calculated using packet-in requests sent from switches to controllers in a per-flow rate manner. In this study, the performance evaluation metrics used for comparison are controller throughput, controller response time, and packet losses. Based on these performance evaluation metrics, the proposed ISMM is compared with the existing one i.e. SSMS and CAMD.

- ***Throughput:*** It is the number of bits received in a given period. It is the sum of tasks completed in one unit of time after performing load balancing. It determines the rate at which the computational work is done using the ISMM. The proposed method aims to achieve greater efficiency. The higher throughput, the better system performance. As expressed in equation 4.1, throughput is the summation of requests that are transmitted from source to destination at a given period, t.

$$\text{Throughput} = \sum_{request\ i}^{n}(time, t) \qquad \textit{Equation 4.1}$$

- ***Packet Loss:*** It is the number of packet losses that happen during sending packets from switches to the controller. When network switches are busy processing incoming packets, they will likely drop some of the incoming packets. The ISMM aims at achieving minimum packet loss to show better controller performance. Therefore, the packet loss can be calculated using the equation 4.2:

$$P(loss) = \frac{Tx\ packets - Rx\ packet}{Tx\ packets} \qquad \textit{Equation 4.2}$$

Where $T_{x\ packet:}$ Total number of transferred packets in OpenFlow switches, $R_{x\ packet:}$ Total number of the packet that is as received, and $P_{(loss):}$ Packet Loss

- ***Response Time:*** It is defined as the summation of the time request takes for the user to retrieve the results of a request. The ISMM aims at achieving the smallest controller response time to show better controller performance.

It is estimated that when there is a controller load imbalance, the response time in a given network will be maximized. For response time calculation, this thesis work adopts equation 4.2 developed by(Yoshizawa, n.d.). The ISMM aims at achieving the smallest response time to show better controller performance. All variables, in this equation, were fixed during the assessment. The accepted equation is written as:

$$R = Rd + Rt \qquad \textit{Equation 4.3}$$

Equation: Where,

$$\boldsymbol{Rd = 2[D + Cc + Ct\ ] + \left[\ D + \frac{[Cc+Cs]}{2}\ \right]\frac{T-2}{m} + Dln\left[\frac{T-2}{m} + 1\right] + KT\left(\frac{L}{1-L}\right)} \qquad \textit{Equation 4.4}$$

$$\boldsymbol{Rt = \frac{MAX\left[8p\frac{1+OHD}{B} * D\frac{P}{W}\right]}{1-\sqrt{L}}} \qquad \textit{Equation 4.5}$$

In each equation, where **P:** payload length, **OHD:** overhead ratio, **B:** minimum path bandwidth and **W:** effective window size, **R**: response time, $\mathbf{R_d}$: propagation delay time, $\mathbf{R_t}$**:** transmission delay time, **D**: round-trip delay, $\mathbf{C_c}$**:** current processing time, **m**: multiplexing factor, $\mathbf{Dl_n}$**:** packet-loss ratio, **K:** TCP timeout, $\mathbf{C_t}$**:** server TCP processing, $\mathbf{C_s}$**:** server processing time, **T**: application turns.

To simulate the proposed solution into the original SDN controller, the below simulation setup was used. Mininet Simulator was used to develop our proposed method, including updating the existing module of ISMM and improving the existing works by modifying their existing algorithm.

4.5.2. Simulation Setup

To simulate the proposed solution the following simulation setup was used to show the new method improvement. The experiment has been conducted on the Mininet simulation tool with java based ODL controller which supports OpenFlow v1.3. The general simulation setup and parameter used are listed in table 4.1.

Table 4.1 Simulation setup parameters

Parameters description	**Value**
OpenFlow Protocol	OF v1.3
Simulation Tool	Mininet 2.3.0
Controllers	ODL
Number of switches	12
Number of controllers	4
Controller Threshold	1800 pps
Total incoming load capacity	10,000-25,000 kbps
Mice flow	1 pps – 499 pps
Elephant flow	501 pps – 5010 pps
Experiment Iteration Executed times	1501 times
Performance Evaluation Metrics	Throughput, Response time, and Packet loss

4.6. Simulation Result

In this thesis, the Mininet simulator is used to perform simulation for obtaining the preferred set of results. A Mininet simulator is a tool that supports research in SDN. It is a python-based simulator and runs on any platform i.e. Linux, and windows.

When the number of the controller is single (one) they have several limitations such as unable to adapt to traffic changes and SPoF. Figure 4.3 shows a single with four switches.

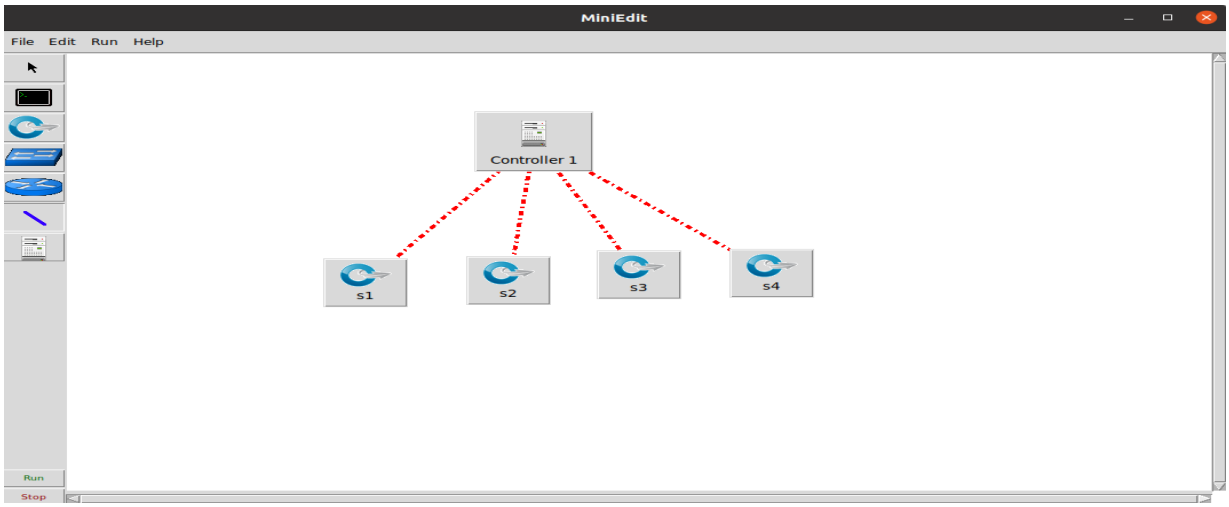


Figure 4.3 Single Network Topology in MiniEdit

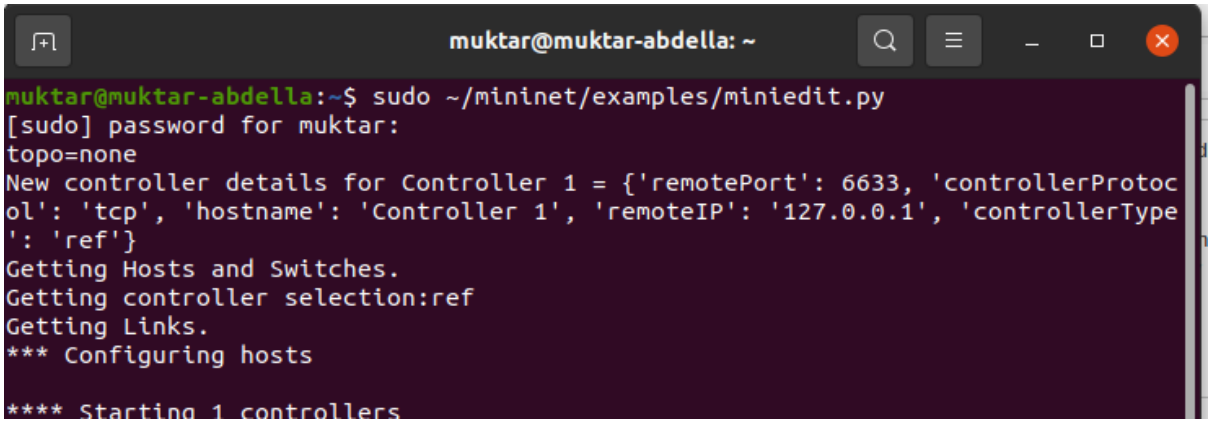


Figure 4.4 Single Controller Execution

To solve the problem of single controller multiple controllers is used in our work. We run our implementation on the Mininet simulator which has four controllers and twelve switches. The C_1 contains 3 switches, C_2 contains 4 switches, C_3 contains 2 switches and C_4 contains 3 switches that are designed on MiniEdit GUI. Based on our investigation C_2 is overloaded and C_3 is underloaded, so there is a need to migrate the switches with the highest load to underloaded controller C_3. Through the simulation run, the controller threshold is set to initial and dynamically updated if needed based on the controller's bandwidth, memory, and CPU. The traffic loads between 501pps- 5010pps were used in studies that act as elephant flows in network topology because it improves the overall throughput and brings more stable balanced networks than mice flow which is between 1pps-499pps.

The reason why we use four controllers and twelve switches in our design topology was to create a robust control topology that is error-free and strong, to make simply interconnections between all the network elements, to minimize the cost of migration, to minimize migration time between controllers, to validate the strength and topology adaptability of ISMM. The network topologies are designed in MiniEdit GUI with the port number 6633, controller type OpenFlow reference, and remote IP address 127.0.01. After configuration, we fill the capacity with a different number of packet-in messages. The switch in this topology communicates with four controllers to balance the load between controllers. The load in multiple controller topology is balanced in a better way. Lastly, the designed topology is run and get the result in appendix H.

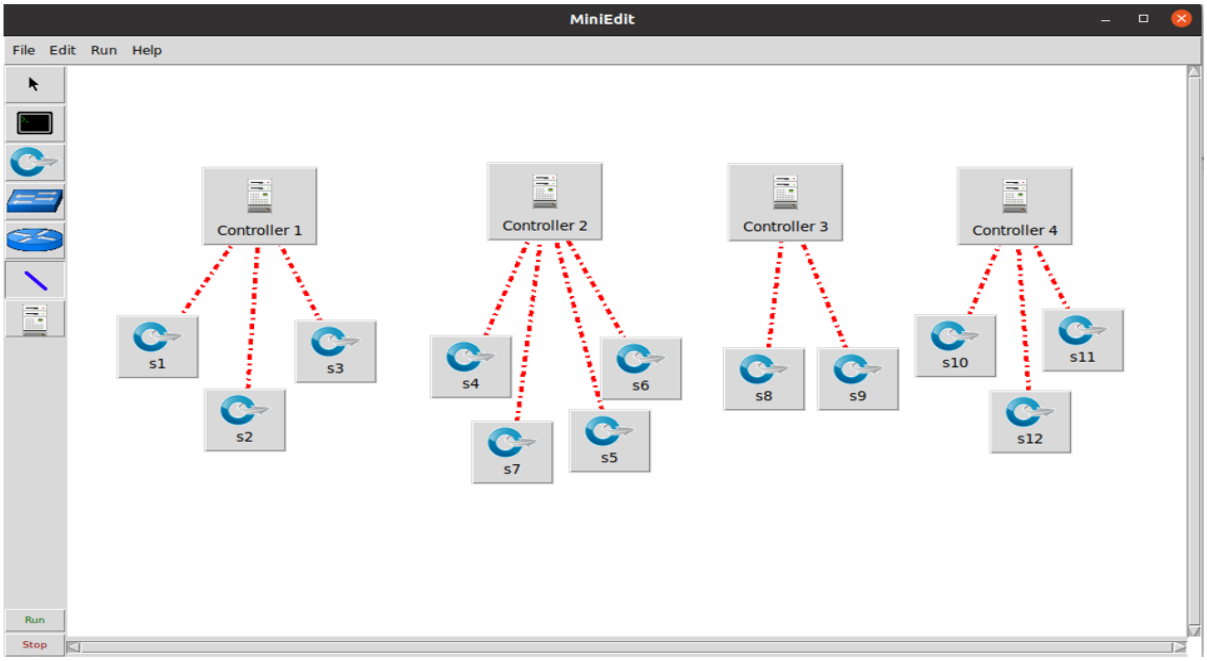


Figure 4.5 Network topology in MiniEdit GUI

Figure 4.5 shows the remote IP, ControllerType, remote port, Protocol of each controller after running on MiniEdit GUI. The proposed model for the starting 4 controllers. The capacity of each controller is equal, but the load or packet-in message sent from switches to each controller is differ because traffic varies from time to time. After 4 controllers start initial controller threshold is set as 1800 pps and the experiments execute 1501 times.

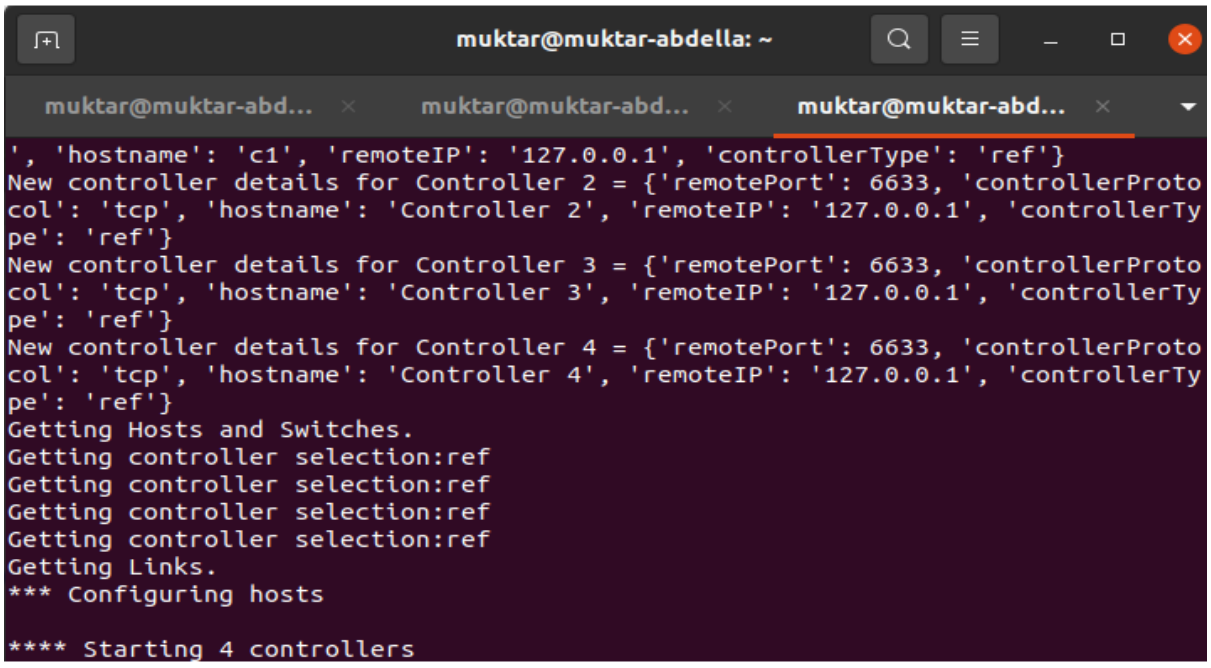


Figure 4.6 Sample of 4 Controllers Execution

Another issue is the reason of metrics selection for the study. Controller throughput, response time, and packet loss are used based on the gaps of related works and to improve existing works. As the number of controller increases cost of configuration is also increase so we limit our design to 4 controllers and twelve switches in our works.

4.7. Analysis of Simulation Result

The simulation graphs show the comparison of the SSMS and CAMD with the proposed ISMM. CAMD is the least loaded controller selection from the underloaded controller that may only be efficient in accepting load shifting when the incoming traffic flow is mice flow. SSMS is also the least loaded controller selection which optimal migrates objects from the underloaded controller for accepting load shifting that used two-stage to perform the migration. This method is also not efficient under elephant traffic flow. The below comparison of three methods was measured based on four controllers by setting the threshold for each controller and dynamically updating the controllers' threshold based on the computational capacity of the controllers. The total in-coming loads for each controller are between 10,000kbps-25,000kbps and the initial controller threshold is 1800pps. More of their comparison result was described in the below section.

4.7.1. Comparison of Result in Terms of Throughput

Figure 4.7 depicts the simulation results of the controller throughputs of each of the reviewed works and the proposed ISMM that shows the maximum amount of packets that can be processed successfully by a controller at a given period. In our simulation, the controller throughput was predictable based on the traffic generated during the simulation process. The total incoming load used in this work was between 10,000kbps – 25,000kbps and the average throughput was set between 390pps - 450 pps. The result shows the controller throughput assessments with SSMS and CAMD and proposed ISMM receive more traffic than CAMD and SSMS methods. The controller throughput of the proposed solution in figure 4.5 increases by approximately 7.6% over the SSMS and about 1.8% over the CAMD. So, there is a big difference between the throughput of SSMS, CAMD, and ISMM results. This indicates that the proposed ISMM maximizes throughput under balanced load distributions.

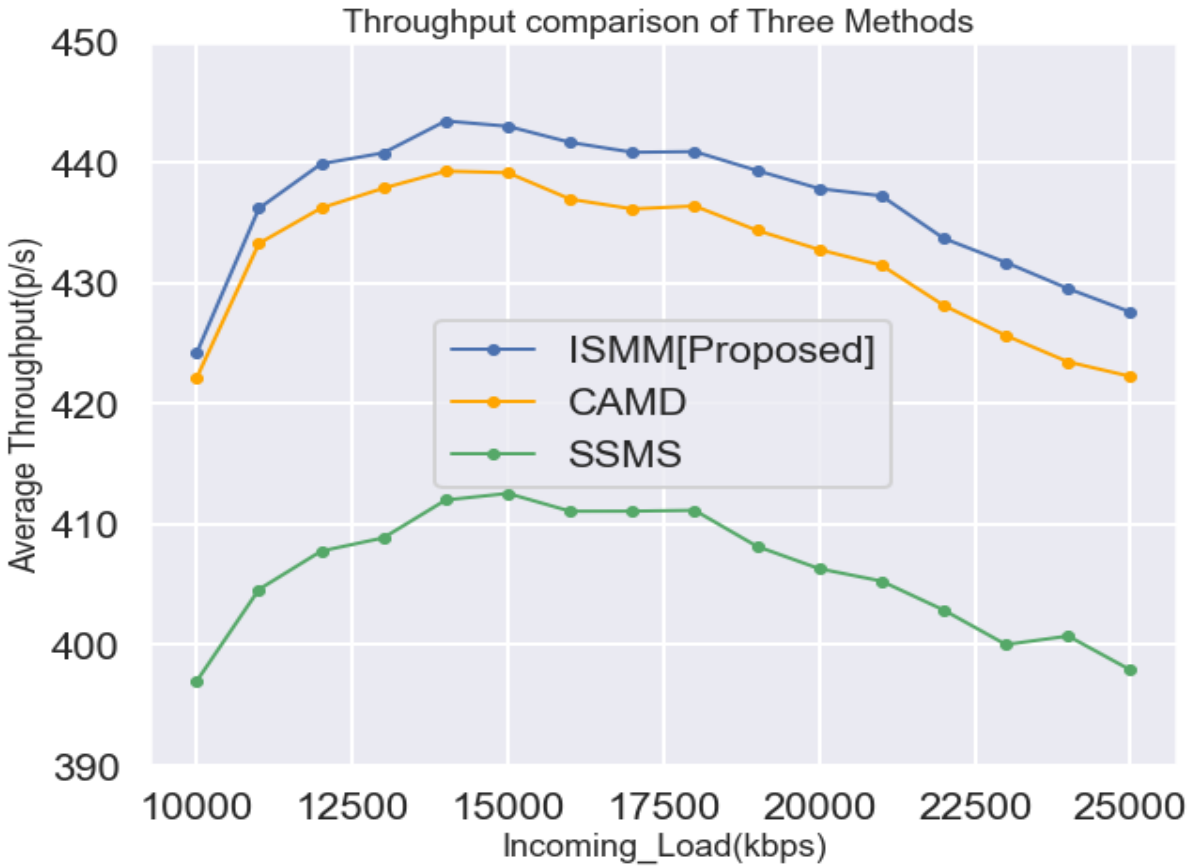


Figure 4.7 Throughput Comparison of the Three Methods

4.7.2. Comparison of Results in terms of Packet Loss

Figure 4. 8 shows the result of the packet loss simulation. The total incoming load used in this work was between 10,000kbps – 25,000kbps and the average packet loss was between (1-16000) %. As incoming load increases, the average packet lost increases for the three methods as described in the figure. However, SSMS had the highest average packet loss when compared with ISMM and CAMD had little improvement in average packet loss compared with ISMM. The analysis assessed that the ISMM has 8% improvement over SSMS and 1.3% improvement over CAMD. This is so due to the effective method taken in this study. Accordingly, it minimizes the number of packets loss in the proposed ISMM. So, the gap raised in the related work was filled by minimizing the number of packet losses in our works.

To fix the pps range from 501-5010 in packet loss, packet loss metric is used to test the packet loss in % and the test mode is used an incremental number of packets sent. The test starts with sending 501pps and then, increase by 2000pps until the maximum of 5010pps. Finally, as

shown in the figure below the number packet reject is minimized using our works, even if the number packet is lower than or above the elephant flow ranges used in this study.

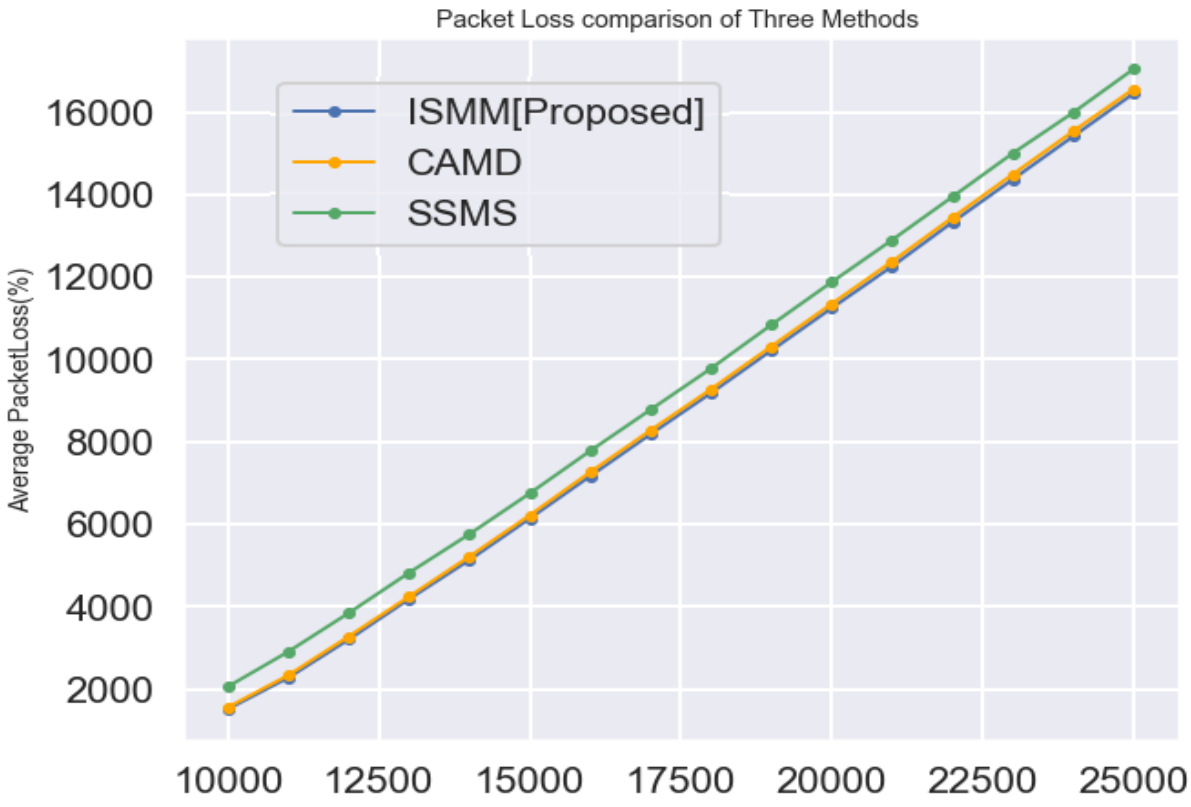


Figure 4.8 Packet Loss comparison of the Three Methods

4.7.3. Comparison of Result in Terms of Response Time

Figure 4.9 reveals the response time of the simulated three methods. The total incoming load used in this study was between 10,000kbps – 25,000kbps and the average response time was between (150100– 550100) measured in milliseconds (ms). ISMM outperforms better improvement over response time also. Consequently, ISMM indicates that resources were well consumed and significantly reduce the controller response time. The controller response time of the proposed solution in figure 4.9 increases by approximately 8.3% over the SSMS and about 4% over the CAMD. Generally, ISMM reduces the response time of the requests than the other two methods to achieve its objective and fill the gap in related works.

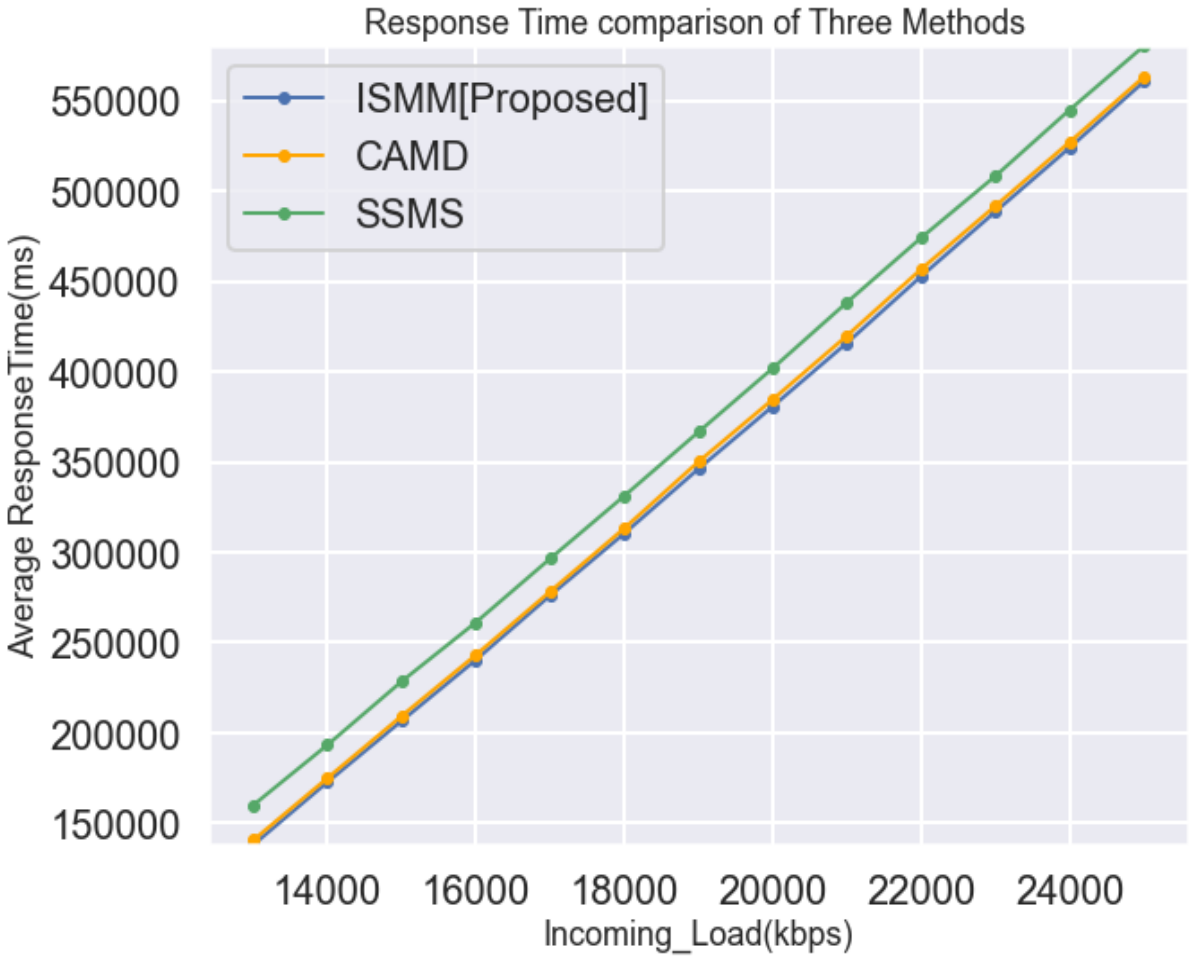


Figure 4.9 Response Time comparison of the Three Methods

4.8. Result Evaluation

The performance evaluation for the three methods is described generally in table 4.2, and the results are taken by executing the simulation 1501 times to obtain the accuracy of the experiment. The result information is taken from the simulation result directory file **C:\Users\Muktar A\Desktop\Implementation*result*** and the MiniEdit GUI. Based on the result it assistances to conclude that the proposed ISMM improved over two methods and improves the performance of the controller by balancing the load between them. The result of all performance evaluation metrics in the proposed ISMM improved over SSMS and CAMD by minimizing the number of packet losses, minimizing response time, and maximizing the controller throughput. This due happens due to their balanced load distribution and dynamically connection of traffic loads. All those verified he proposed ISMM performs better in all measuring procedures.

Therefore, this study used a simulation approach and the assessment of throughput, response time, and packet loss for the performance result assessment. The current load on every 4 controllers before any migration period was set to be 501pps means that the initial controller's load. The total incoming load was between 10000pps – 26000pps, while each of these iterations was executed 1501 times to obtain the accuracy of the experiment. The threshold of the controller is set as 1800pps and it is updated dynamically based on a load of a controller. The reason why we take this threshold was based on initial packet-in requests sent to controllers. All this taken information is taken for justifying our result and performed for each controller in the network domain.

The ISMM will run all four modules on each controller to balance the load between each controller. First, the load measurement and judgment module are started for the computation of the load. If the load computed load is more than the predefined controller threshold, the controller is overloaded and dynamically updates the controller threshold. If controller load is less than controller threshold, subsequently, the ISMM will start the switch selection module and the controller selection module simultaneously, then perform the switch migration to balance the load. The switch with the highest load is selected for migration and the most appropriate controller is also selected for accepting the incoming load. Finally, ISMM improved over other methods.

Figure 4.10 shows the throughput results for the three methods by considering their average throughput between (390pps-450pps) and the incoming-load between (10000kbps -25010kbps) which shows improvement than the two other methods.

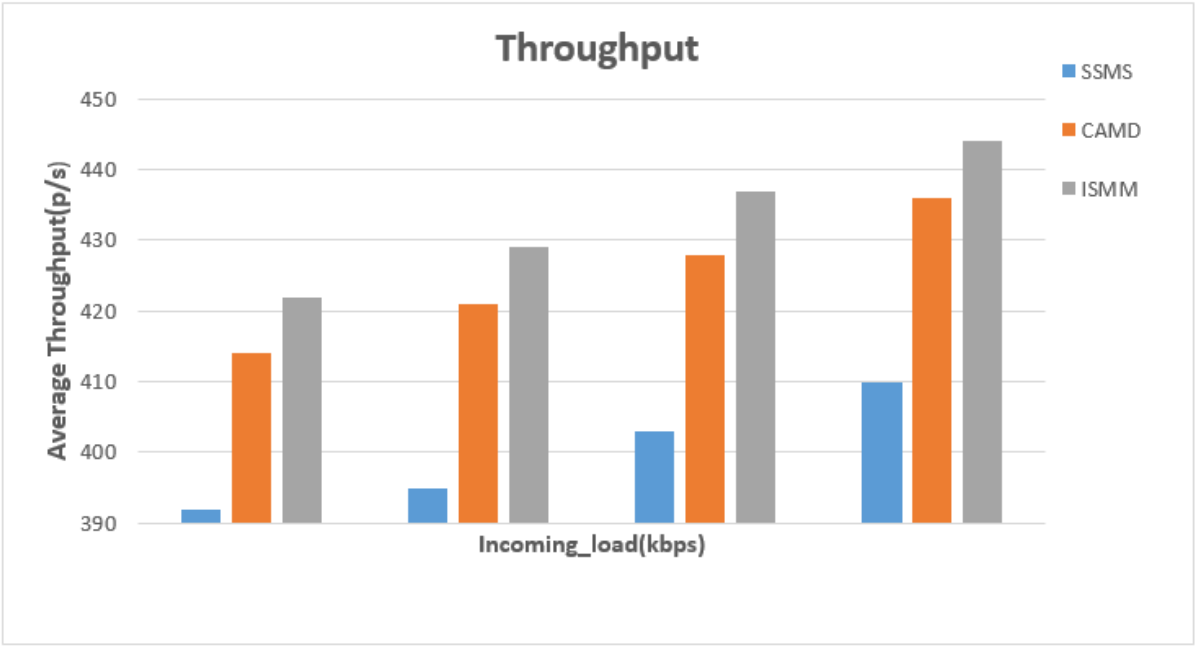


Figure 4.10 Throughput Comparison of Three Method

Figure 4.11 shows the packet loss for three methods by considering average packet loss (0-16000) and the incoming load 10000kbps -25010kbps. In this case, the result reveals that the ISMM outperforms the most efficient result in terms of packet loss. Finally, the number packet loss is minimized as shown in table 4.2.

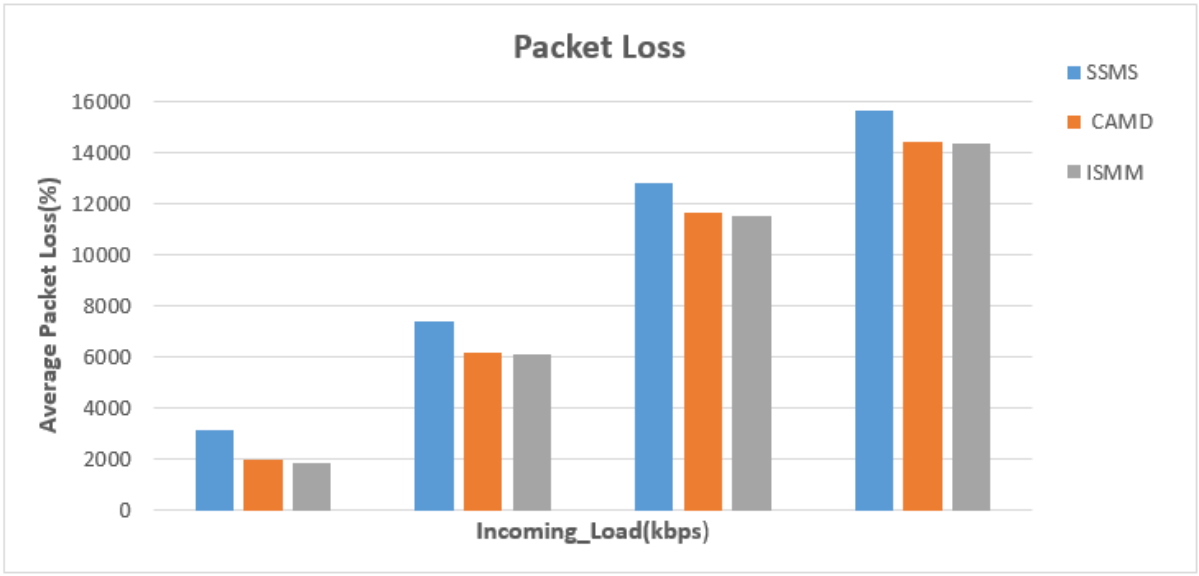


Figure 4.11 Packet Loss Comparison of Three Method

Figure 4.12 shows the controller response time for three methods by considering average response time (ms) and the load incoming (kbps). In this case, ISMM performs in the most

efficient result when compared with two other methods. CAMD in this evaluation of result is almost approach to SSMS but to some extent improved by ISMM.

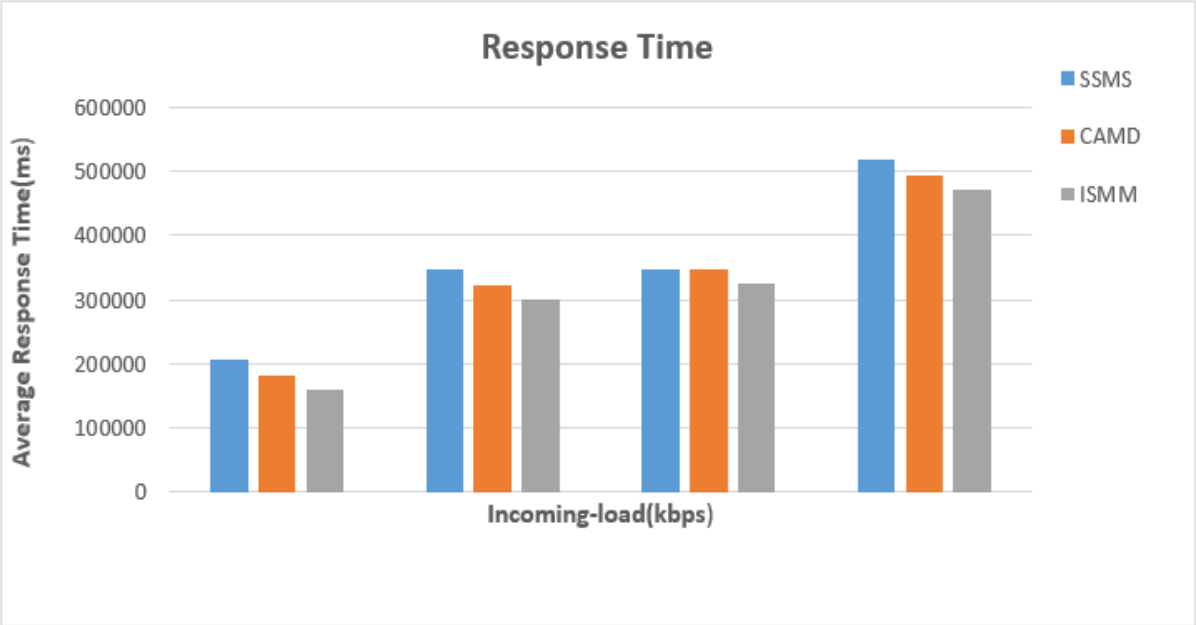


Figure 4.12 Response Time comparison of Three Method

Therefore, our study answers all the raised question in research and meet the specific objective by improving the existing method which improves three metrics in our work. Based on the above result in figures 4.10, 4.11, and 4.12, we conclude that the proposed method ISMM outperforms better results over SSMS and CAMD. The implementation of a switch migration method can migrate a switch from high loaded to an underloaded one based on controller threshold capacity.

Result Summary: As described in the above sections three methods are simulated using a similar environment such as number of controllers, number of switches, controller thresholds, simulation time, and load incoming capacity. The proposed ISMM was efficient than CAMD and SSMS in all measured metrics. When one controller is overloaded, it migrates the switches of its domain that have the highest load to the most appropriate underloaded controller. So, the load is balanced and the proposed solution gives better controller performance. In table 4.2 the individual improvements of the three methods are described based on the figure of results comparison.

Table 4.2 Summary of Three Method Evaluation

Average Throughput(pps)	**Throughput of Three Method**		
	SSMS	**CAMD**	**ISMM**
	↑ 392 pps	↑ 414 pps	↑ 422 pps
	↑ 395 pps	↑ 421 pps	↑ 429 pps
	↑ 403 pps	↑ 428 pps	↑ 437 pps
	↑ 410 pps	↑ 436 pps	↑ 444 pps
Packet Loss of Three Method			
Average Packet Loss (%)	**SSMS**	**CAMD**	**ISMM**
	↓ 3171	↓1981	↓ 1891
	↓ 7367	↓ 6167	↓ 6087
	↓ 12820	↓ 11620	↓ 11540
	↓ 15600	↓ 14400	1 4320
Response Time of Three Method			
Average Response Time(ms)	**SSMS**	**CAMD**	**ISMM**
	↓ 206600 ms	↓ 182600 ms	↓ 160400 ms
	↓ 346400 ms	↓ 322400 ms	↓ 300200 ms
	↓ 348600 ms	↓ 346400 ms	↓ 324000 ms
	↓ 518000 ms	↓ 494000 ms	↓ 471800 ms

This table is generated from the comparison results to verify that the proposed ISMM was effective in balanced loads and improving the overall performance of the networks.

Generally, figure 4.13 describes the summarization of the above table percentage values. The graph shows by what percent does the related works and proposed ISMM increase the controller performance and decrease the controller loads between four controllers in our implementation. As the graph shows there was improvement between the existing CAMD, SSMS, and the proposed ISMM in three performance metrics. The graph describes that as the number of load incoming increases all three performance metrics decreased. But, this issue was handled by our proposed ISMM, in this method, the overloaded controller migrates the

switch of its domain to the underloaded controller if it surpasses the maximum thresholds. To do this migration our proposed ISMM uses a four-module includes a load judgment and measurement module, switch selection module, a controller selection module, and a switch migration module.

Throughput: In our study, the proposed ISMM improves throughput by 7.6% over SSMS and 1.8% over CAMD. This indicates the improvement of the ISMM over the two methods. Finally, the proposed ISMM maximizes the throughput of controllers.

Response time: In our study, the proposed ISMM improves response time by 8.3% over SSMS and 4% over CAMD. This indicates the improvement of the ISMM over the two methods. Finally, the proposed ISMM minimizes controller response time.

Packet loss: In our study, the proposed ISMM improves packet loss by 8% over SSMS and 1.3% over CAMD. This indicates the improvement of the ISMM over the two methods. Finally, the proposed ISMM minimizes the number of packet losses in our study.

Table 4.3 Performance Improvement of the Proposed Method

Metrics Improved	SSMS Improvement by %	CAMD Improvement by %
Throughput	7.6%	1.8%
Response Time	8.3%	4%
Packet Loss	8%	1.3%

Therefore, ISMM improves all three metrics with the best efficiency in comparison with SSMS and CAMD is shown in table 4.3 and the graph result shown in figure 4.13 is generated from table 4.2.

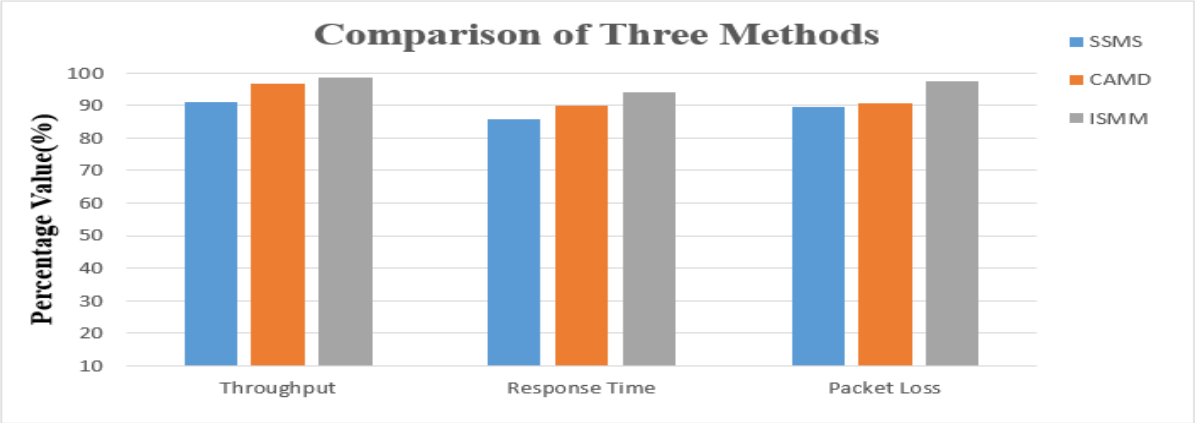


Figure 4.13 Comparison of Three Methods Summary

4.9. Discussion

The evaluation of the result simulated on the Mininet simulation tool cross-checked with other methods indicates that the ISMM is the best method for solving the controller load imbalance problem. The proposed ISMM work based on the four-module includes load measurement and judgment module, switch selection module, appropriate controller selection module, and switch migration module. For validating the performance of this study, the ISMM is compared with other similar works in the related work. This study used throughput, response time, and packet loss for the performance evaluation. The ISMM shows improvement in terms of throughput, packet loss, and response time which are shown in table 4.2 confirms that the proposed ISMM has the best result in all measured metrics. So, our proposed ISMM solves the controller load imbalance problems and brings significant improvement in terms of metrics evaluated.

The four stated research questions in this study were addressed based upon the result, and performance evaluation metrics have been done. It is discussed as the following:

The first question raises the effect of controller load imbalance in the controller performance. As discussed in the related work, the existing methods affect the performance of the system since it does not equally distribute the incoming load among the controllers and is inefficient under high traffic flow. Due to this, the throughput is minimized, response time is maximized, and packet loss is maximized under the unbalanced load of the controller. Therefore, this study answers the first research question by improving all three metrics, to show its improvement over two other methods. Finally, the throughput is maximized, response time is minimized, and packet loss is also minimized in our experiment.

The second research question raised in this study is answered by adjusting controller loads dynamically if they all controller exceed the threshold. So, the proposed ISMM answers this question and shows the improvement of this method.

The third research question raises an improved existing load balancing method. In this study, this question is answered by improving three metrics and the result is shown in table 4.3 and figure 3.13.

The fourth question raises the way of evaluation of the performance of the proposed solution. Here, the performance of the proposed ISMM is evaluated by considering three metrics for performance assessment. The proposed ISMM is compared with SSMS and CAMD methods.

4.10. Validation of the Switch Selection and Controller Selection

This study illustrates a simple example to prove the significance of load variance and migration efficiency in switch migration conditions under the same controller capacity. Assuming four controllers, C1, C2, C3, C4, and their switch load is denoted as in table 4.4. Controller threshold is assumed as 1800pps

Table 4.4 Switch to controller Packet-in Flow Rate

Controller set	**C_1**			**C_2**				**C_3**		**C_4**		
Switch set	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S112
Switch load (pps)	400	300	300	300	500	400	700	200	300	400	300	300
T	**1800pps**											

4.10.1. Switch Selection Illustration Example

In this study, we look at a switch with maximum load flow requests to the controller that is overloaded during the migration phase as this affects the network load balancing rate (LBR). Table 4.4, shows the packet-in rate of the flow from various switches to the controller. Controller C_2, at the time (t), has a total load of 1900pps which makes it an overloaded controller. Controller C_3, with higher migration efficiency, will be selected as the target controller for load shifting. So, we describe how the LBR is affected by choosing a switch with fewer flow requests. This study had considered equation 3.6 for the evaluation of LBR which is defined in the equation below.

$$\boldsymbol{LBR} = (\frac{1}{n}\sum_{i=1}^{n}(Ci))/\ \underset{i=1}{max}\, n\,(Ci)$$

Where $\{C_i \ldots C_n\}$ constitutes the load list of the connected systems which includes the overloaded controller's load within the controller domain. Before migration, the LBR is computed as 0.5789. For instance, if controller C_2, selects switch S_4, its lightest loaded switch for migration, the LBR in equation 3.6 after migration will be 0.6875. Meanwhile, if C_2 decides to select its highest loaded switch S_7 for migration, the LBR becomes 0.92. From the above analysis, this study revealed that selecting a switch with the highest flow request will bring about significant improvement in network efficiency.

4.10.2. Controller Selection Illustration Example

In this study, load variance is used to identify the most appropriate controller from the list of the under-loaded controller. Assuming the controller C= (C_1, C_2, C_3, C_4) and their load is represented as LS= (1000, 1900, 500, 1000) pps. The average load of the controller LS= 1100pps. The controller threshold is assumed as 1800pps. Due to the above computation, C2 is an overloaded controller because it exceeds the controller threshold. Generally, the load variance determines the deviation of the set of numbers from its load mean value, in a given distribution. The individual controller variance are (C_1)= 10,000, $\boldsymbol{\nu}$(C_2)= 640,000 $\boldsymbol{\nu}$(C_3)= 360,000 $\boldsymbol{\nu}$(C_3)=10,000 and the variance of the load of the controller before migration is 255,000.

The overloaded controller C_2 can either shift extra load to C_3 and C_1 or C_4. For instance, if it chooses C_1 or C_4, the individual variance and its global variance for the network are lower. Subsequently, if C_3 is chosen the individual variance and its global variance are higher. Therefore, the migration efficiency of C_3 is higher than the C_1 or C_4-based individual variance of the controller. Therefore, this study concluded that considering a higher migration efficiency controller is the best choice. C_3 will be considered as the target controller because of the most appropriate controller and the overall load balance will be in a stable mode.

4.11. Summary

In this chapter, the controller load imbalance issue is solved by using ISMM and evaluated based on three metrics. To solve this problem different simulations were done. The performance of SSMS, CAMD and proposed ISMM were assessed with performance metrics of the throughput, response time, and packet loss with the same simulation environment, and simulations were executed 1501 times to obtain the accuracy of the experiment. Based on the result we have discussed load balance between multiple controllers from the perspective of three methods. Then, the evaluation of results shows migrating switch from the loaded controller to the underloaded controller using ISMM is better. Balancing controller load among multiple controllers is done for better performance improvement by migrating the switch from one controller to other controllers if it surpasses the controller threshold.

We have improved the metrics such as throughput, response time, and packet loss. Finally, the controller throughput is maximized, controller response time is minimized and the number of packet losses is minimized using ISMM. The improvement of the proposed solution is shown with the comparison with the SSMS and CAMD methods from the perspective of three metrics. Lastly, the improvement of the method, and how the researcher answers the research question in our study are discussed in detail. So, the proposed ISMM method achieves better improvement under three evaluated metrics. The conclusion, output of the study, and future work was discussed in the next chapter.

CHAPTER FIVE

5. CONCLUSION AND FUTURE WORK

5.1. Conclusions

In our study, the problem of controller load imbalance between multiple controllers in SDN is occurred due to the unbalanced load distribution among the controllers and due to static mapping between switch and controller. This makes some controllers overload and other underloaded. Different kinds of literature were reviewed to find a better solution for the problems. To meet our objective ISMM is proposed. To balance this load, switching is migrated from an overloaded one to an underloaded one is a possible solution. The proposed ISMM used four modules including load measurement and judgment module, switch selection module, a controller selection module, and switch migration module.

The design of ISMM runs locally on each controller. The load measurement and judgment module in this design check immediately whether the controller load surpasses the predefined threshold and make decisions. Each controller in the network design collects all controller loads and calculates the controller's load mean. This is used to decide the set of underloaded controllers and overloaded controllers in the network topology. If the current load on the controller is above a predefined threshold it's overloaded and if the current load on the controller is below a predefined threshold it's underloaded. Secondly, the switch selection module is also becomes activated which is used to select the highest incoming loaded switch from the set of overloaded controllers for the migration procedure which improves the selection efficiency. Thirdly, the controller selection module also gets started beside modules 1 and 2 to ensure the most suitable controller from a set of underloaded controllers is selected. Finally, the fourth module accomplishes switch migration and balance loads between controllers.

The proposed ISMM simulated on Mininet Simulation tools in our study, since it’s freely open-source, has better computation cost, popular to solve the problems of load imbalance, and supports OpenFlow protocol. ODL controller is used because it supports a network of any size and scale, modular, and robust.

Therefore, we conclude that based on the simulation conducted, the result is revealed that the proposed ISMM improved over SSMS and CAMD between the traffic loads of 501pps-5010pps, in terms of three performance assessments metrics. Finally, it improves controller throughput, improves the packet loss and response time of the control plane.

5.2. The output of the Study

The output of this study was in the proposed ISMM that balances the load between multiple controllers. ISMM contains modules. Each module of ISMM runs on every four controllers and set controller threshold initial at 1800pps on all of the simulated controllers. Based on this, switches are migrated, if it surpasses the controller threshold that is initially set and the ISMM method runs on each controller to balance controllers load and improve controller performance. Lastly, the proposed ISMM is simulated on the Mininet simulation tool.

The main contribution of this thesis work are summarized as follows:

- This work studies controller load balancing based upon the switch migration.
- This study improves the existing switch migration work and proposed ISMM to balance controller loads between multiple controllers.
- The ISMM achieved better performance improvement in terms of throughput, response time, packet loss, and the result compared with SSMS and CAMD methods to show improvement of works. This improvement is achieved because the effective method is brought forward in the study. We modify on the existing algorithms.
- When the controller becomes overloaded, the proposed ISMM used the mean load to identify the set of the underloaded controller and overloaded controller and simultaneously involves the switch migration module.
- In this work, the switch is selected based on LBR, due to the degree of closeness to the real load distribution can be measured with LBR.
- In this work, to select the target controller from the set of underloaded controller load variance has been formulated, and most appropriate controller selected based on variance.
- In this work, the controller threshold is updated dynamically, if all controller load exceeds the initial threshold.
- Finally, the proposed ISMM has been implemented using Mininet simulator with ODL controller on designed network topology and compared the result with SSMS and CAMD.

Generally, the proposed solution improves controller throughput by 7.6% over SSMS and 1.8% over CAMD. The controller response time is improved by 8.3% over SSMS and 4% over CAMD. The packet loss is improved by 8% over SSMS and 1.3% over CAMD. Generally, this indicates the improvement of the proposed ISMM over two methods, and the proposed method improves the performance of the controller.

5.3. Future Work

Due to time constraints, this work considers three performance evaluation metrics. So, we recommend the future researcher consider other metrics such as migration time, migration cost, and degree of load balancing. This work was done on simulation, so it's better to implement it in a real network environment in the future. Another issue that needed to be considered for further study is controller failure scenario, controller energy consumption, security scenario, reliability, and scalability.

Finally, our recommendation is to be used this works in different companies such as telecom, cloud data center, home network, campus network, ISP is to use SDN.

SPECIAL ACKNOWLEDGMENT

This research work was funded by Adama Science and Technology University under grant number:

ASTU/SM-R/239/21

Adama, Ethiopia

REFERENCES

Abdelaziz, A., Fong, A. T., Gani, A., Garba, U., Khan, S., Akhunzada, A., Talebian, H., & Choo, K. K. R. (2017). Distributed controller clustering in software defined networks. *PLoS ONE*, *12*(4), 0–19. https://doi.org/10.1371/journal.pone.0174715

Abdullah, M. Z., Al-awad, N. A., & Hussein, F. W. (2018). *Performance Evaluation and Comparison of Software Defined Networks Controllers*. *November*. https://doi.org/10.12785/IJCNT/060201

Adekoya, O., Aneiba, A., & Patwary, M. (2020). An Improved Switch Migration Decision Algorithm for SDN Load Balancing. *IEEE Open Journal of the Communications Society*, *1*, 1602–1613. https://doi.org/10.1109/ojcoms.2020.3028971

Al-Tam, F., & Correia, N. (2019). On load balancing via switch migration in software-defined networking. *IEEE Access*, *7*, 95998–96010. https://doi.org/10.1109/ACCESS.2019.2929651

AL-Tam, F., & Correia, N. (2019). Fractional switch migration in multi-controller software-defined networking. *Computer Networks*, *157*(April), 1–10. https://doi.org/10.1016/j.comnet.2019.04.011

Babbar, H., Rani, S., Masud, M., Verma, S., Anand, D., & Jhanjhi, N. (2021). Load Balancing Algorithm for Migrating Switches in Software-Defined Vehicular Networks. *Computers, Materials and Continua*, *67*(1), 1301–1316. https://doi.org/10.32604/cmc.2021.014627

Bannour, F., Souihi, S., & Mellouk, A. (2018). Distributed SDN Control: Survey, Taxonomy, and Challenges. *IEEE Communications Surveys and Tutorials*, *20*(1), 333–354. https://doi.org/10.1109/COMST.2017.2782482

Beiruti, M. A., & Ganjali, Y. (2019). Migration scheduling in distributed SDN controllers. *Proceedings - International Conference on Network Protocols, ICNP*, *2019-Octob*, 1–2. https://doi.org/10.1109/ICNP.2019.8888045

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

Benzekki, K., El Fergougui, A., & Elbelrhiti Elalaoui, A. (2016). Software-defined networking

(SDN): a survey. *Security and Communication Networks*, *9*(18), 5803–5833. https://doi.org/10.1002/sec.1737

Blial, O., Ben Mamoun, M., & Benaini, R. (2016). An Overview on SDN Architectures with Multiple Controllers. *Journal of Computer Networks and Communications*, *2016*. https://doi.org/10.1155/2016/9396525

Chandra, S., Srujan, S. K., Krishna, K. S. D. R., & Editors, M. N. F. (n.d.). *Learning and Analytics in Intelligent Systems 3 Advances in Decision Sciences , Image Processing , Security and Computer Vision* (Vol. 1).

Eissa, H. A., Bozed, K. A., & Younis, H. (2019). Software Defined Networking. *19th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering, STA 2019, November*, 620–625. https://doi.org/10.1109/STA.2019.8717234

EstiNet - Simulator | EstiNet. (n.d.). Retrieved September 15, 2021, from https://www.estinet.com/ns/

Filali, A., Cherkaoui, S., & Kobbane, A. (2019). Prediction-Based Switch Migration Scheduling for SDN Load Balancing. *IEEE International Conference on Communications*, *2019-May*(July). https://doi.org/10.1109/ICC.2019.8761469

Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D. C., & Gayraud, T. (2014). Software-defined networking: Challenges and research opportunities for future internet. *Computer Networks*, *75*(PartA), 453–471. https://doi.org/10.1016/j.comnet.2014.10.015

Hamdan, M., Hassan, E., Abdelaziz, A., Elhigazi, A., Mohammed, B., Khan, S., Vasilakos, A. V., & Marsono, M. N. (2021). A comprehensive survey of load balancing techniques in software-defined network. *Journal of Network and Computer Applications*, *174*. https://doi.org/10.1016/j.jnca.2020.102856

Hock, D., Hartmann, M., Gebert, S., Jarschel, M., Zinner, T., & Tran-Gia, P. (2013). Pareto-optimal resilient controller placement in SDN-based core networks. *Proceedings of the 2013 25th International Teletraffic Congress, ITC 2013*. https://doi.org/10.1109/ITC.2013.6662939

Hu, T., Guo, Z., Yi, P., Baker, T., & Lan, J. (2018a). Multi-controller Based Software-Defined

Networking: A Survey. *IEEE Access*, *6*(c), 15980–15996. https://doi.org/10.1109/ACCESS.2018.2814738

Hu, T., Guo, Z., Yi, P., Baker, T., & Lan, J. (2018b). Multi-controller Based Software-Defined Networking: A Survey. *IEEE Access*, *6*, 15980–15996. https://doi.org/10.1109/ACCESS.2018.2814738

Hu, T., Lan, J., Zhang, J., & Zhao, W. (2019). EASM: Efficiency-aware switch migration for balancing controller loads in software-defined networking. *Peer-to-Peer Networking and Applications*, *12*(2), 452–464. https://doi.org/10.1007/s12083-018-0632-6

Hu, T., Yi, P., Zhang, J., & Lan, J. (2018). A distributed decision mechanism for controller load balancing based on switch migration in SDN. *China Communications*, *15*(10), 129–142. https://doi.org/10.1109/CC.2018.8485475

Hu, T., Yi, P., Zhang, J., Lan, J., Zhang, S., Lan, J., Sun, P., Jiang, Y., Hu, T., Zhang, J., Wang, L., Qiao, D., Babbar, H., Rani, S., Masud, M., Verma, S., Anand, D., Jhanjhi, N., LI, J., … Choo, K. K. R. (2019). A Switch Migration-Based Decision-Making Scheme for Balancing Load in SDN. *IEEE Access*, *5*(4), 4537–4544. https://doi.org/10.1109/ACCESS.2017.2684188

Hu, T., Zhang, J., Wang, L., & Qiao, D. (2017). *An Improved Switch Migration Approach to Controller Load Balancing in SDN*. *124*(Isaeece), 436–442. https://doi.org/10.2991/isaeece-17.2017.83

Jadhav, K. A., Mulla, M. M., & Narayan, D. G. (2020). An Efficient Load Balancing Mechanism in Software Defined Networks. *Proceedings - 2020 12th International Conference on Computational Intelligence and Communication Networks, CICN 2020*, 116–122. https://doi.org/10.1109/CICN49253.2020.9242601

Jammal, M., Singh, T., Shami, A., Asal, R., & Li, Y. (2014). Software defined networking: State of the art and research challenges. *Computer Networks*, *72*, 74–98. https://doi.org/10.1016/j.comnet.2014.07.004

Kaur, P., Chahal, J. K., & Bhandari, A. (2018). Load Balancing in Software Defined Networking: A Review. *Asian Journal of Computer Science and Technology*, *7*(1), 1–5.

Kawale, N., Srinivas, L. N. B., & Venkatesh, K. (2021). Review on traffic engineering and

load balancing techniques in software defined networking. In *Lecture Notes in Networks and Systems* (Vol. 130). Springer Singapore. https://doi.org/10.1007/978-981-15-5329-5_18

Killi, B. P. R., & Rao, S. V. (2019). Controller placement in software defined networks: A Comprehensive survey. *Computer Networks*, *163*. https://doi.org/10.1016/j.comnet.2019.106883

Kreutz, D., Ramos, F. M. V., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, *103*(1), 14–76. https://doi.org/10.1109/JPROC.2014.2371999

Li, G., Li, K., Liu, Y., & Pan, Y. (2019). An efficient dynamic load balancing scheme based on nash bargaining in SDN. *Future Internet*, *11*(12). https://doi.org/10.3390/FI11120252

LI, J., SUN, E., & ZHANG, Y. (2018). Multi-Threshold SDN Controllers Load Balancing Algorithm Based On Controller Load. *DEStech Transactions on Computer Science and Engineering*, *CCNT*, 398–407. https://doi.org/10.12783/dtcse/ccnt2018/24732

Lin, W., & Zhang, L. (2016). *The Load Balancing Research of SDN based on Ant Colony Algorithm with Job Classification*. *Wartia*, 472–476. https://doi.org/10.2991/wartia-16.2016.95

Ma, Y. W., Chen, J. L., Tsai, Y. H., Cheng, K. H., & Hung, W. C. (2017). Load-Balancing Multiple Controllers Mechanism for Software-Defined Networking. *Wireless Personal Communications*, *94*(4), 3549–3574. https://doi.org/10.1007/s11277-016-3790-y

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

Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet. (n.d.). Retrieved September 15, 2021, from http://mininet.org/

Mostafavi, S., & Hakami, V. (2019). Performance Evaluation of Software-Defined Networking Controllers : A Comparative Study. *Journal of Computer and Knowledge Engineering*, *2*(2), 63–73. https://doi.org/10.22067/cke.v2i2.84917

Muluye, W. (2020). A Review on Software-Defined Networking Distributed Controllers.

International Journal of Engineering and Computer Science, *9*(2), 24953–24961. https://doi.org/10.18535/ijecs/v9i2.4439

Neghabi, A. A., Navimipour, N. J., Hosseinzadeh, M., & Rezaee, A. (2018). Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature. *IEEE Access*, *6*, 14159–14178. https://doi.org/10.1109/ACCESS.2018.2805842

ns-3 | a discrete-event network simulator for internet systems. (n.d.). Retrieved September 15, 2021, from https://www.nsnam.org/

Paliwal, M., Shrimankar, D., & Tembhurne, O. (2018). Controllers in SDN: A review report. *IEEE Access*, *6*(March 2011), 36256–36270. https://doi.org/10.1109/ACCESS.2018.2846236

Sahoo, K. S., Puthal, D., Tiwary, M., Usman, M., Sahoo, B., Wen, Z., Sahoo, B. P. S., & Ranjan, R. (2020). ESMLB: Efficient Switch Migration-Based Load Balancing for Multicontroller SDN in IoT. *IEEE Internet of Things Journal*, *7*(7), 5852–5860. https://doi.org/10.1109/JIOT.2019.2952527

Sahoo, K. S., & Sahoo, B. (2019). CAMD: A switch migration based load balancing framework for software defined networks. *IET Networks*, *8*(4), 264–271. https://doi.org/10.1049/iet-net.2018.5166

Saraswat, S., Agarwal, V., Gupta, H. P., Mishra, R., Gupta, A., & Dutta, T. (2019). Challenges and solutions in Software Defined Networking: A survey. *Journal of Network and Computer Applications*, *141*(April), 23–58. https://doi.org/10.1016/j.jnca.2019.04.020

Semong, T., Maupong, T., Anokye, S., Kehulakae, K., Dimakatso, S., Boipelo, G., & Sarefo, S. (2020). Intelligent load balancing techniques in software defined networks: A survey. *Electronics (Switzerland)*, *9*(7), 1–24. https://doi.org/10.3390/electronics9071091

Sharma, R. (2021). A Review on Software Defined Networking. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, *3307*, 11–14. https://doi.org/10.32628/cseit21728

Simulink Documentation. (n.d.). Retrieved September 15, 2021, from https://www.mathworks.com/help/simulink/

Tiwari, G., Chakaravarthy, V. D., & Rai, A. (2019). Dynamic load balancing in software defined networking. *International Journal of Engineering and Advanced Technology*, *8*(5), 2706–2712.

Vasudevan, D., & Nayak, S. (2018). Software-Defined Networks. *IEEE Potentials*, *37*(5), 21–24. https://doi.org/10.1109/MPOT.2015.2448733

Wang, C., Hu, B., Chen, S., Li, D., & Liu, B. (2017). A Switch Migration-Based Decision-Making Scheme for Balancing Load in SDN. *IEEE Access*, *5*(c), 4537–4544. https://doi.org/10.1109/ACCESS.2017.2684188

Wang, H., Xu, H., Huang, L., Wang, J., & Yang, X. (2018). Load-balancing routing in software defined networks with multiple controllers. *Computer Networks*, *141*, 82–91. https://doi.org/10.1016/j.comnet.2018.05.012

Wang, Y., Yu, J., Pei, K., & Qiu, X. (2017). A Load Informing Based Load Balancing Mechanism for Multiple Controllers in SDN. *Dianzi Yu Xinxi Xuebao/Journal of Electronics and Information Technology*, *39*(11), 2733–2740. https://doi.org/10.11999/JEIT161054

Wickboldt, J., De Jesus, W., Isolani, P., Both, C., Rochol, J., & Granville, L. (2015). Software-defined networking: Management requirements and challenges. *IEEE Communications Magazine*, *53*(1), 278–285. https://doi.org/10.1109/MCOM.2015.7010546

Xu, Y., Cello, M., Wang, I. C., Walid, A., Wilfong, G., Wen, C. H. P., Marchese, M., & Chao, J. H. (2019). Dynamic Switch Migration in Distributed Software-Defined Networks to Achieve Controller Load Balance. *IEEE Journal on Selected Areas in Communications*, *37*(3), 515–529. https://doi.org/10.1109/JSAC.2019.2894237

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

Yao, L., Hu, T., & Lan, J. (2018). Sequenced Switch migration for balancing controller loads in large-scale software-defined networking. *International Journal of Performability Engineering*, *14*(8), 1848–1856. https://doi.org/10.23940/ijpe.18.08.p22.18481856

Yeo, S., Naing, Y., Kim, T., & Oh, S. (2021). Achieving balanced load distribution with

reinforcement learning-based switch migration in distributed SDN controllers. *Electronics (Switzerland)*, *10*(2), 1–16. https://doi.org/10.3390/electronics10020162

Yoshizawa, S. (n.d.). *Performance Prediction Method for Web-*. 905–906.

Zhou, Yang, Zheng, K., Ni, W., & Liu, R. P. (2018). Elastic Switch Migration for Control Plane Load Balancing in SDN. *IEEE Access*, *6*(c), 3909–3919. https://doi.org/10.1109/ACCESS.2018.2795576

Zhou, Yuanhao, Zhu, M., Xiao, L., Ruan, L., Duan, W., Li, D., Liu, R., & Zhu, M. (2015). A load balancing strategy of SDN controller based on distributed decision. *Proceedings - 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2014*, *978*, 851–856. https://doi.org/10.1109/TrustCom.2014.112

Zhu, L., Sharif, K., & Du, X. (2019). *SDN Controllers : Benchmarking & Performance Evaluation*. *December*.

APPENDIXES

Appendix A: Installation of Mininet and ODL Controller

We can simply install Mininet on Ubuntu 20.04.2 LTS using the command:

>Sudo apt-get install Mininet.

We can also install the controller by using the command on Ubuntu 20.4.2 LTS:

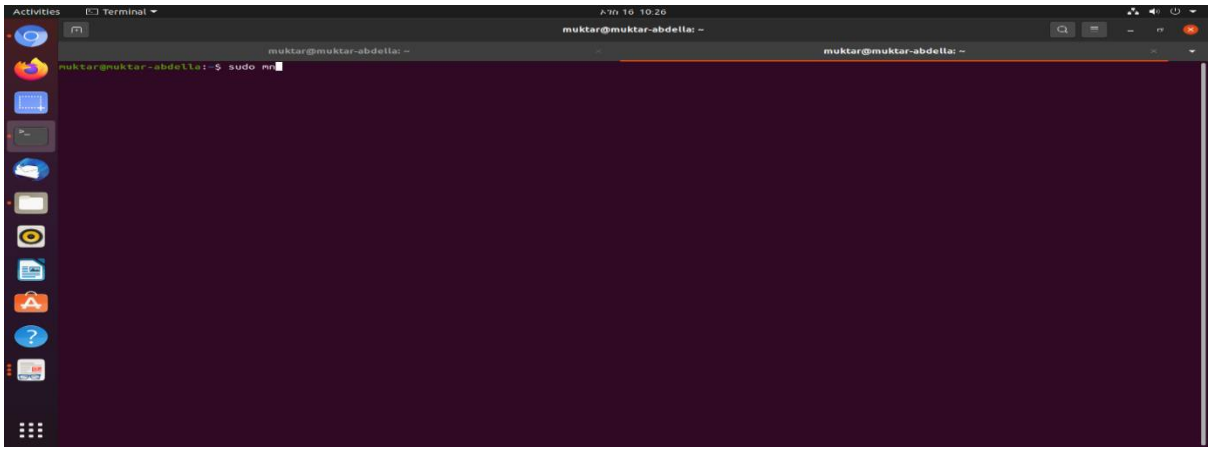
>Sudo apt-get install OpenDayLight.

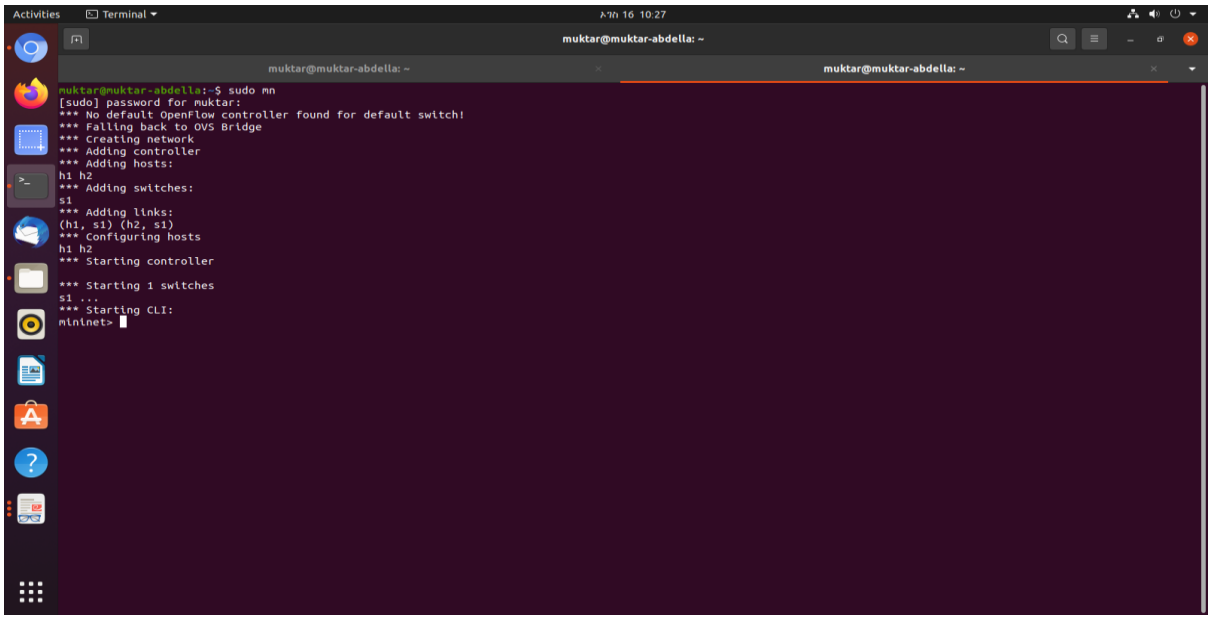
Appendix B: Basic Commands in Mininet.

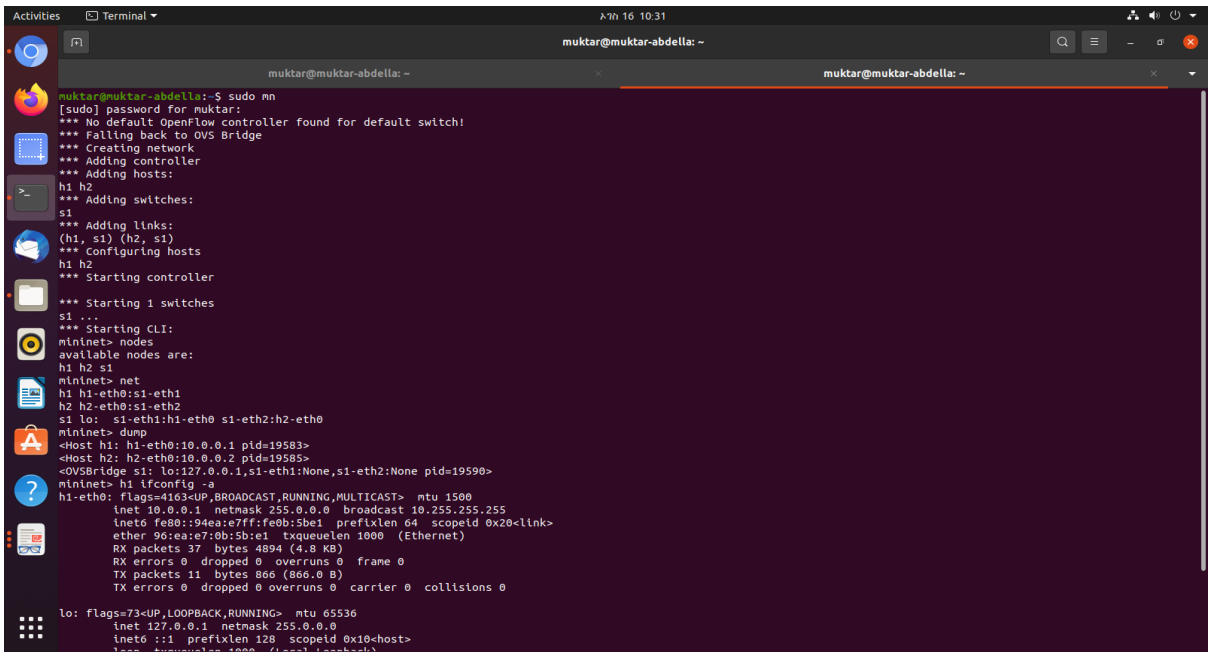
We use the following comment in Mininet while using Mininet simulation tools in our implementation.

- ***$:*** It precedes commands that are typed at shell prompt.
- ***Mininet>:*** It is commanded for leading to Mininet CLI.
- ***Sudo mn:*** to start default topology of Mininet interface.
- ***Sudo mn –h:*** we use the command sudo mn–h to see the help menu.
- ***Sudo mn –c:*** we can use the command to clear Mininet.
- ***Mininet>help:*** we use this command to show several options on Mininet CLI.
- ***Mininet>nodes:*** we use this command to show available nodes in the current network.
- ***Mininet>dump:*** we use this command to show the dump information about all available nodes in the current Mininet network.
- ***Mininet> h1 ping h2:*** we use this command to check the connectivity between host 1 and host 2. This is done until we break the command.
- ***Mininet> h1 ping -c1 h2:*** we use this command to check the connection between h1 and h2.
- ***Mininet> h1 ifconfig -a:*** we use this command to show the h1-eth0 and loopback.
- ***Mininet> s1 ifconfig -a:*** is the command to run switch by default in the network as a root.
- ***Mininet>pingall:*** we use this command to check the connectivity among all hosts.

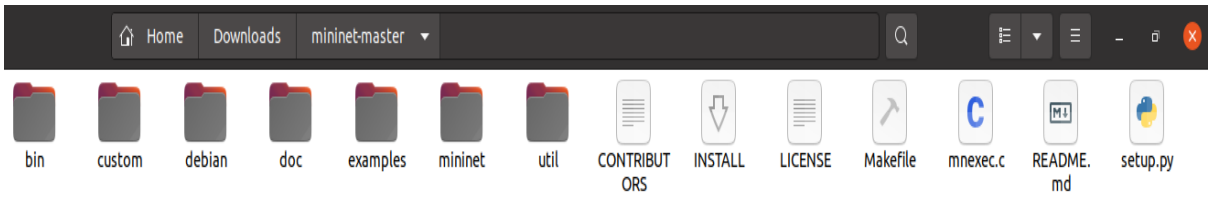
Appendix C: Startup Command Prompt for Mininet Simulator

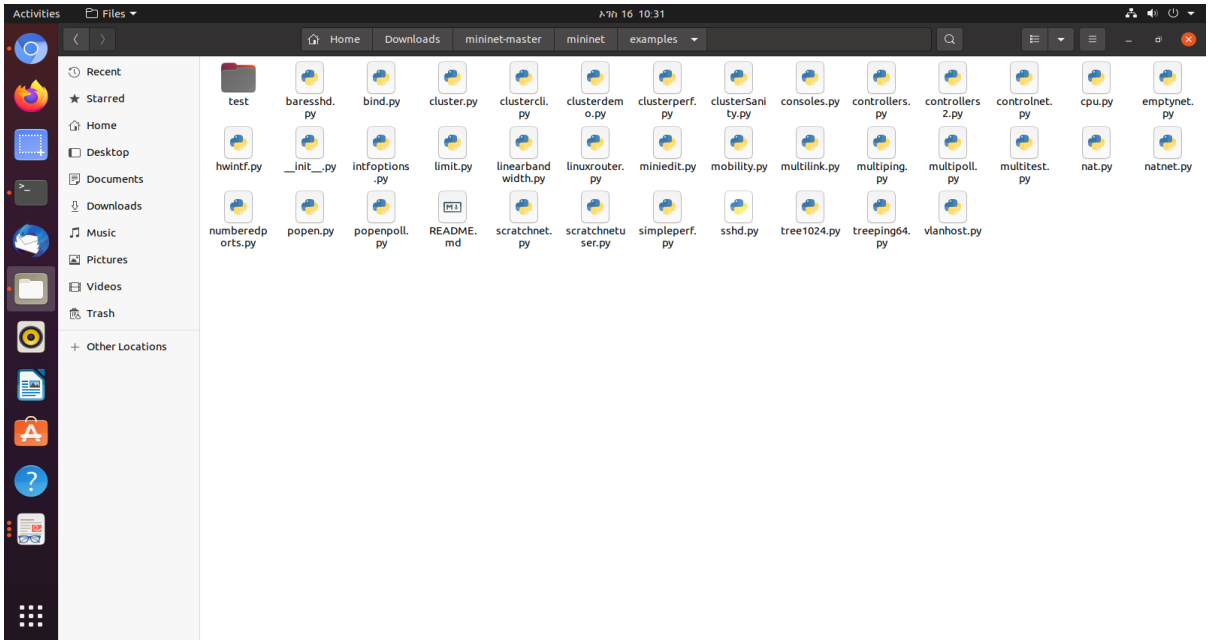






Appendix D: Mininet Simulation Tools Directory





Appendix E: Sample code for Designed Topology

```
from mininet.net import Mininet
from mininet.node import OVSSwitch, Controller, RemoteController, Node
from mininet.topolib import TreeTopo
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.topo import Topo
IPBASE='172.20.0.'
NATOMIX=2
class MySwitch( OVSSwitch ):
    global myCmap
    "Custom Switch() subclass that connects to different controllers"
    def __init__(self, *args, **kwargs):
        OVSSwitch.__init__(self, *args, **kwargs)
        self.start([ myCmap[ self.name ] ])
class MyTopo(Topo):
    def __init__(self,**kwargs):
        super(MyTopo, self ).__init__(self,**kwargs)
        s1=self.addSwitch('s1', cls=MySwitch)
        s2=self.addSwitch('s2', cls=MySwitch)
```

```
        s3=self.addSwitch('s3', cls=MySwitch)
        s4=self.addSwitch('s4', cls=MySwitch)
        s5=self.addSwitch('s5', cls=MySwitch)
        s6=self.addSwitch('s6', cls=MySwitch)
        s7=self.addSwitch('s7', cls=MySwitch)
        s8=self.addSwitch('s8', cls=MySwitch)
        s9=self.addSwitch('s9', cls=MySwitch)
        s10=self.addSwitch('s10', cls=MySwitch)
        s11=self.addSwitch('s11', cls=MySwitch)
        s12=self.addSwitch('s12', cls=MySwitch)
        self.addLink(s1,s2)
        self.addLink(s1,s3)
setLogLevel( 'info' )
c1 = RemoteController( 'c1', ip=IPBASE+str(NATOMIX+2), port=8181 )
c2 = RemoteController( 'c2', ip=IPBASE+str(NATOMIX+3), port=8181 )
c3 = RemoteController( 'c3', ip=IPBASE+str(NATOMIX+4), port=8181 )
c4 = RemoteController( 'c4', ip=IPBASE+str(NATOMIX+5), port=8181 )
global myCmap
myCmap = { 's1': c1, 's2': c1,'s3': c1,'s4': c2,'s5': c2,'s6': c2,'s7':
c2, 's8': c3,'s9': c3,'s10': c4,'s11': c4,'s12': c4 }
net = Mininet( topo=MyTopo(), switch=MySwitch, build=False,
waitConnected=True )
for c in [ c1, c2, c3, c4 ]:
    net.addController(c)
net.build()
net.start()
CLI( net )
net.stop()
```

Appendix F: Code Structure for Three Methods

Method 1: SSMS

The least controller selection approach and is represented in the code as "N".

```
for L in incoming_load:
        if method == "N":
            if current_A + L < CT:
                current_1 = max(0, current_A + L - pr_for_A)
                current_B = max(current_B - pr_for_B, 0)
                current_C = max(current_C - pr_for_C, 0)
                current_D = max(current_D - pr_for_D, 0)
                accepted_packets.append(L)
                result_A.append(current_A)
                result_B.append(current_B)
                result_C.append(current_C)
                result_D.append(current_D)
            elif current_B + L < CT:
                current_A = max(0, current_A - pr_for_A)
                current_B = max(current_B + L - pr_for_B, 0)
                current_C = max(current_C - pr_for_C, 0)
                current_D = max(current_D - pr_for_D, 0)
                accepted_packets.append(L)

                result_A.append(current_A)
                result_B.append(current_B)
```

```
                result_C.append(current_C)
                result_D.append(current_D)

            elif current_C + L < CT:
                current_A = max(0, current_A - pr_for_A)
                current_B = max(current_B - pr_for_B, 0)
                current_C = max(current_C + L - pr_for_C, 0)
                current_D = max(current_D - pr_for_D, 0)
                accepted_packets.append(L)

                result_A.append(current_A)
                result_B.append(current_B)
                result_C.append(current_C)
                result_D.append(current_D)

            elif current_D + L < CT:
                current_A = max(0, current_A - pr_for_A)
                current_B = max(current_B - pr_for_B, 0)
                current_C = max(current_C - pr_for_C, 0)
                current_D = max(current_D + L - pr_for_D, 0)
                accepted_packets.append(L)

                result_A.append(current_A)
                result_B.append(current_B)
                result_C.append(current_C)
                result_D.append(current_D)

            else:
                rejected_packets.append(L)
                current_A = max(current_A - pr_for_A, 0)
                current_B = max(current_B - pr_for_B, 0)
                current_C = max(current_C - pr_for_C, 0)
                current_D = max(current_D - pr_for_D, 0)

                result_A.append(current_A)
                result_B.append(current_B)
                result_C.append(current_C)
                result_D.append(current_D)
```

Methods 2: CAMD

Least loaded controller selector approaches which is represented in the code as "L".

```
elif method == "L":
   UT = [{"Name": "A", "Currentload": current_A}, {"Name": "B",
"Currentload": current_B}, {"Name": "C", "Currentload": current_C},
{"Name": "D", "Currentload": current_D} ]
   UT.sort(key=lambda x: x["Currentload"], reverse=False)
   UT = [controller for controller in UT if controller["Currentload"] + L
<= CT]
            if len(UT) != 0:
                if UT[0]['Name'] == "A":
                    current_A = max(0, current_A + L - pr_for_A)
                    current_B = max(current_B - pr_for_B, 0)
                    current_C = max(current_C - pr_for_C, 0)
                    current_D = max(current_D - pr_for_D, 0)
```

```
                accepted_packets.append(L)

                result_A.append(current_A)
                result_B.append(current_B)
                result_C.append(current_C)
                result_D.append(current_D)
            elif UT[0]['Name'] == "B":
                current_B = max(0, current_B + L - pr_for_B)
                current_A = max(current_A - pr_for_A, 0)
                current_C = max(current_C - pr_for_C, 0)
                current_D = max(current_D - pr_for_D, 0)
                accepted_packets.append(L)

                result_A.append(current_A)
                result_B.append(current_B)
                result_C.append(current_C)
                result_D.append(current_D)
            elif UT[0]['Name'] == "C":
                current_C = max(0, current_C + L - pr_for_C)
                current_A = max(current_A - pr_for_A, 0)
                current_B = max(current_B - pr_for_B, 0)
                current_D = max(current_D - pr_for_D, 0)
                accepted_packets.append(L)

                result_A.append(current_A)
                result_B.append(current_B)
                result_C.append(current_C)
                result_D.append(current_D)

            elif UT[0]['Name'] == "D":
                current_C = max(0, current_C - pr_for_C)
                current_A = max(current_A - pr_for_A, 0)
                current_B = max(current_B - pr_for_B, 0)
                current_D = max(current_D + L - pr_for_D, 0)
                accepted_packets.append(L)

                result_A.append(current_A)
                result_B.append(current_B)
                result_C.append(current_C)
                result_D.append(current_D)

        else:
            rejected_packets.append(L)

            current_A = max(current_A - pr_for_A, 0)
            current_B = max(current_B - pr_for_B, 0)
            current_C = max(current_C - pr_for_C, 0)
            current_D = max(current_D - pr_for_D, 0)

            result_A.append(current_A)
            result_B.append(current_B)
            result_C.append(current_C)
            result_D.append(current_D)
```

Method 3: ISMM (Proposed)

This proposed method the most appropriate controller selection based approach on the incoming traffic. It is represented as "M" in the code.

```
elif method == "M":
            UT = [{"Name": "A", "Currentload": current_A}, {"Name": "B",
"Currentload": current_B},{"Name": "C", "Currentload": current_C},
{"Name": "D", "Currentload": current_D}]
            UT.sort(key=lambda x: x["Currentload"], reverse=True)
            UT = [controller for controller in UT if
controller["Currentload"] + L <= CT]

            if len(UT) != 0:
                if UT[0]['Name'] == "A":
                    current_A = max(0, current_A + L - pr_for_A)
                    current_B = max(current_B - pr_for_B, 0)
                    current_C = max(current_C - pr_for_C, 0)
                    current_D = max(current_D - pr_for_D, 0)

                    accepted_packets.append(L)

                    result_A.append(current_A)
                    result_B.append(current_B)
                    result_C.append(current_C)
                    result_D.append(current_D)
                elif UT[0]['Name'] == "B":
                    current_B = max(0, current_B + L - pr_for_B)
                    current_A = max(current_A - pr_for_A, 0)
                    current_C = max(current_C - pr_for_C, 0)
                    current_D = max(current_D - pr_for_D, 0)
                    accepted_packets.append(L)

                    result_A.append(current_A)
                    result_B.append(current_B)
                    result_C.append(current_C)
                    result_D.append(current_D)
                elif UT[0]['Name'] == "C":
                    current_C = max(0, current_C + L - pr_for_C)
                    current_A = max(current_A - pr_for_A, 0)
                    current_B = max(current_B - pr_for_B, 0)
                    current_D = max(current_D - pr_for_D, 0)
                    accepted_packets.append(L)

                    result_A.append(current_A)
                    result_B.append(current_B)
                    result_C.append(current_C)
                    result_D.append(current_D)

                elif UT[0]['Name'] == "D":
                    current_C = max(0, current_C - pr_for_C)
                    current_A = max(current_A - pr_for_A, 0)
                    current_B = max(current_B - pr_for_B, 0)
                    current_D = max(current_D + L - pr_for_D, 0)
                    accepted_packets.append(L)
```

```
                    result_A.append(current_A)
                    result_B.append(current_B)
                    result_C.append(current_C)
                    result_D.append(current_D)
            else:
                rejected_packets.append(L)

                current_A = max(current_A - pr_for_A, 0)
                current_B = max(current_B - pr_for_B, 0)
                current_C = max(current_C - pr_for_C, 0)
                current_D = max(current_D - pr_for_D, 0)

                result_A.append(current_A)
                result_B.append(current_B)
                result_C.append(current_C)
                result_D.append(current_D)
```

Appendix G: Sample Code for Changing Roles.

```
route('OurController',
config.CONTROLLER['METHODS']['START_MIGRATION'][0],
methods=[config.CONTROLLER['METHODS']['START_MIGRATION'][1]])
    def start_migrate(self, req, **kwargs):
        # Start Migrating
        jsonData = {
            'sourceController': req.json['source'],
            'targetController': req.json['dest'],
            'targetSwitch': req.json['switch']
        }
        print jsonData
        self.controller.migrationState = 1
        self.controller.migrationData = jsonData

        url           =           jsonData['targetController']           +
str(config.CONTROLLER['METHODS']['MIGRATION_BEGIN'][0])
        util.Http_Request(url, jsonData)
        print "start migration, for switch: " + jsonData['targetSwitch'] +
" from controller: " + jsonData['sourceController'] + " to controller: " +
jsonData['targetController']
        return Response(content_type='text/plain', body='helloworld')
    @route('OurController',
config.CONTROLLER['METHODS']['MIGRATION_BEGIN'][0],
methods=[config.CONTROLLER['METHODS']['MIGRATION_BEGIN'][1]])
    def migration_begin(self, req, **kwargs):
        jsonData = req.json
        dpid = jsonData['targetSwitch']
        self.controller.migrationState = 1
        self.controller.migrationData = jsonData

        datapath = self.controller.switches[int(dpid)]
        print "set role: Equal for dpid " + str(dpid)
        ofp = datapath.ofproto
        ofp_parser = datapath.ofproto_parser
```

```
        req = ofp_parser.OFPRoleRequest(datapath, ofp.OFPCR_ROLE_EQUAL, 0)
# the role request to become equal
        datapath.send_msg(req)
        return
Response(content_type='text/plain', body='migration_begin')
Constant = {
    'Role':{
        'Equal':ofproto.ofproto_v1_3.OFPCR_ROLE_EQUAL,
        'Master':ofproto.ofproto_v1_3.OFPCR_ROLE_MASTER,
        'Slave':ofproto.ofproto_v1_3.OFPCR_ROLE_SLAVE
    }
}
def get_role(i):
    if i == "s":
        return Constant['Role']['Slave']
    elif i == "m":
        return Constant['Role']['Master']
    elif i == "e":
    return Constant['Role']['Equal']
```

Appendix H: MiniEdit GUI in Mininet

The Mininet network simulator used in this implementation contains MiniEdit a simple GUI editor located in file director ***E:\mininet-master\examples\miniedit.py***. We use MiniEdit to design network topology, save the topology, and run the network simulation. To run MiniEdit, execute the command: **$sudo ~/ *mininet-master* /examples/miniedit.py.**