Switch Migration Based Load Balancing for Software Defined Network Multiple Controllers Using Reinforcement Learning

Ayano Shibire Dido

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

September 2023

Adama, Ethiopia

Switch Migration Based Load Balancing for Software Defined Network Multiple Controllers Using Reinforcement Learning

Ayano Shibire Dido

Advisor: Dr. Ketema Adere (PhD)

A Thesis Submitted to the Department of Computer Science and Engineering

School of Electrical 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

September 2023

Adama, Ethiopia

Declaration

I declare that this Master Thesis entitled "**Switch Migration Based Load Balancing for Software Defined Network Multiple Controllers Using Reinforcement Learning"** is my own work and has not been submitted to any university for a similar purpose. The references used in this proposal are duly recognized by proper citations.

Ayano Shibire	_____	_____
Name of student	Signature	Date

Recommendation of Advisor

I, the major advisor of this thesis, hereby certify that I have read the revised version of the thesis entitled "**Switch Migration Based Load Balancing for Software Defined Network Multiple Controllers Using Reinforcement Learning**" prepared under my guidance by **Ayano Shibire Dido** submitted in partial fulfillment of the requirements for the degree of Mater's of Science in Computer Science and Engineering. Therefore, I recommend the submission of revised version of the thesis to the department following the applicable procedures.

Dr. Ketema Adere	_____	_____
Major Advisor	Signature	Date

Approval Page

I, the advisor of the thesis "**Switch Migration Based Load Balancing for Software Defined Network Multiple Controllers Using Reinforcement Learning**" and developed by **Ayano Shibire Dido**. 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 **Ayano Shibire Dido** have read and evaluated the thesis "**Switch Migration Based Load Balancing for Software Defined Network Multiple Controllers Using Reinforcement Learning**" 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.

_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
Internal Examiner	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

ACKNOWLEDGMENTS

Foremost, I would like to express my gratitude to the almighty Allah who helped to attain this level through my journey. I would like to express my sincere gratitude to my advisor Dr. Ketema Adere for his continuous support of my MSc study and research work. His guidance helped me in all the time of research work and writing of this thesis. He is one of the mentors I admire the most and I would like to say thank you again.

My sincere thanks also go to my entire family for their support and advice in my life journey. Finally, I wish to thank all my friends who supported me in the completion of my thesis and All Network Science and Security SIG students for their significant contribution, support, comments, and motivation during my work. Especially thanks to Hadi Hussen and Gutema Bote who are former students of Network Science SIG, for their support, comments, and suggestions on my work.

Table of Contents

Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ACRONYMS

A

AI: Artificial Intelligence

API: Application Program Interface

B

BalCon: Balanced Controller

C

CNN: Convolution Neural Network

CAPEX: Capital Expenditure

CV: Cluster Vector

D

DCN: Data Center Network

DDPG: Data Deterministic Policy Gradient

DHA: Distributed Hopping Algorithm

DNN: Deep Neural Network

DRL: Deep Reinforcement Learning

DSMS: Dynamic Switch Migration Strategy

DT: Decision Tree

DQN: Deep Q-Network

E

ECMP: Equal Cost Multi-Path Routing

EMD: Earth Distance Mover's

EASM: Efficiency Aware Switch Migration

EASM: Efficiency Aware Switch Migration

F

FMOPSO: Fuzzy Multi-Objective Particle Swarm Optimization

H

HMM: Hidden Markov Model

HTTP: Hyper Text Transfer Protocol

I

IoT: Internet of Things

IP: Internet Protocol

L

LAN: Local Area Network

LSTM: Long Short-Term Memory

M

MARL: Multi-Agent Reinforcement Learning

ML: Machine Learning

MDP: Markov Decision Process

MRBS: Multiple Regression Based Searching

N

NBI: North Bound Interface

NFV: Network Function Virtualization

NN: Neural Network

O

OPEX: Operational Expenditure

OSPF: Open Shortest Path First

P

POCO: Pareto-based Optimal Controller Placement

R

RBF: Radial Based Function

RL: Reinforcement Learning

RLB: Reinforcement Learning Load Balancing

RMON: Remote Network Monitoring

RQ: Research Question

S

SAR-LB: Switch Aware Reinforcement Learning Load Balancing

SBI: South Bound Interface

SDN: Software Defined Networking

SMP: Switch Migration Problem

SNMP: Simple Network Management Protocol

SOM: Self Organizing Map

SSH: Secure Shell

SSMS: Stated Switch Migration Strategy

SVM: Support Vector Machine

V

VM: Virtual Machine

W

WAN: Wide Area Network

X

XML: Extensible Markup Language

Abstract

SDN is a new computer networking paradigm that is built on the concept of separating the control plane and data plane to simplify network management through central network programmability and quick innovation. Because network traffic fluctuates over time and space, a static mapping between switches and controllers results in unequal load distribution among controllers. Multiple controller deployment in SDN is a viable technique to improve the SDN controller's reliability and scalability. However, it introduces a new issue of load imbalance between multiple controllers due to dynamic network traffic changes and imbalanced incoming load allocation across the controllers. This leads to load imbalance issues between controllers, causing some controllers to become overloaded and some others. Existing dynamic switch migration approaches are mostly focused on load balancing of distributed SDN controllers with switch migration schemes, and they are based on threshold definition for load judgments, which results in migration overhead. We use online Q-learning with the constraints of maximum efficiency and no migration conflicts to obtain global optimal controller load balancing at the lowest cost. Our work resulted in better outcomes in a variety of scenarios with varying load distribution on switches. We demonstrate the convergence behavior of the proposed scheme to the optimal policy using simulation results. The result shows the designed scheme outperforms better in terms of load balancing rate and switch migration exchange and achieves 19% improvements compared to RLB and 12% compared to SAR-LB in load balancing rate. Also resulted in a maximized cumulative reward of the system.

Keywords: *Multiple controllers; software-defined networking (SDN); load balancing; switch migration; Reinforcement learning*

CHAPTER ONE

INTRODUCTION

1.1 Background of the study

The complexity of network infrastructure, resources, and organization has expanded dramatically as Internet technology and mobile communication continue to advance at a rapid pace. Furthermore, the Internet has evolved as a key infrastructure, fueling social development and technological progress while having a significant impact on people's jobs, education, and lifestyles.

The Data Center Network (DCN) experiences growth as a result of increased utilization by Internet applications, including activities such as video streaming, e-commerce, social networking, and the expanding data storage needs of these applications, which can reach up to 40 ZB (Begam et al., 2022b). Even when we consider the usage rate of internet in our country Ethiopia, During the 2020/21 fiscal year, Ethio Telecom in Ethiopia experienced a significant increase in data and internet users, reaching a total of 24.9 million subscribers. This marks a 4.7 percent growth compared to the previous year and represents the highest number of subscribers they have had since the 2012/13 period and There is a growth of 34.5% in voice traffic and 94.5% in data traffic when compared to the 2022 fiscal year (*Ethio Telecom's Data and Internet Users Ethiopia 2021 | Statista*, n.d.). In order to meet the increasing demand for data storage and access, the implementation of effective load balancing techniques is essential. The goal is to reduce latency and response time by selecting the most efficient server and path for each request. This not only maximizes throughput but also minimizes resource usage, ultimately improving user satisfaction.

Traditional network technology, on the other hand, has inherent constraints such as rigid structures and intricate setups that cannot satisfy the demands of network innovation. These classic network architectures have several flaws that limit their capacity to successfully manage large-scale networks, assure dependable network services, administrate network devices, and implement new network protocols. To overcome these difficulties, Software-Defined Networking (SDN) has evolved as a new network design.

SDN is a trendy networking paradigm that makes the network more adaptable by adding programming capability to it. This network customization feature allows infrastructure and operations to better utilize resources based on the needs of the organization

and market demand. End-user devices, storage data centers, and wide area networks are the participants in a software-defined industry that improves network operators' control, programming, and response to business requirements. (Belgaum et al., 2020). The impetus for the transition to SDN and Network Functions Virtualizations (NFV) has resulted in software and virtualization taking advantage of agility, flexibility, and adaptability. This has a significant impact on how networks are planned, managed, and services are delivered. Software, which is becoming a more significant component of any network, accepts changing end-user demands for increased programming and openness. This level of business shift necessitates major network reorganization.

Service traffic in networks is rapidly increasing due to the rapid development of network communication technologies and the large increase in network terminal equipment. Simultaneously, these rapidly expanding service flows pose significant problems to current communication networks. The Internet has undergone a significant transformation as a result of the new intelligent network architecture represented by SDN.

Traditional routing-optimization techniques always use the optimized heuristic algorithms to forward traffic, ECMP, or OSPF. The OSPF protocol directs all flow requests to the shortest path. Due to a lack of intelligence to learn from previous experiences, these strategies may result in the repeating of incorrect conclusions when similar traffic patterns occur. The poor path decision causes network congestion, which leads to further performance degradation and the inability to meet global load-balancing needs.

SDN introduced a new network architecture that is intended to improve on traditional networks. This novel strategy emphasizes centralized management, resulting in lower hardware costs for both CAPEX and OPEX. This design also has remarkable scalability, vendor neutrality, and a focus on empowering operators to manage networks precisely and without errors. It provides flexible network management that reacts to changing network demands easily, enabling a dynamically adjustable network architecture (Hakiri et al., 2014; Jammal et al., 2014). SDN is a revolutionary networking architecture that separates the control plane and the data plane, simplifying and adapting network management. This separation of the control plane and data plane, together with the incorporation of intelligence into the SDN controller, enables networks to become programmable via Application Programming Interfaces (APIs) (Rupani et al., 2020).

Managing, organizing, optimizing, and maintaining network systems demands the study and use of a large amount of data. However, putting machine learning (ML) into normally closed networks presented difficulties. SDN transformed network flexibility, agility, and programmability, allowing researchers to investigate multiple network domains using software-driven solutions. ML is an important technology that can be implemented into SDN design to improve network functionality, addressing both functional and non-functional elements like performance, security, and others (Faezi & Shirmarz, 2023). ML is made up of three important components: the model, which makes predictions or identifications; the parameters, which are signals or features that improve the model's prediction or classification capabilities; and the training data. The SDN has become one of the most popular topics. It is clear that the attachment of ML and SDN has resulted in a lot of research that has improved performance. As a result, the benefit of rapidly obtaining data via SDN, which is the most crucial aspect of ML learning, has become an incentive to draw additional attention (Quach et al., 2020).

Load balancing is the process of dividing workloads among multiple resources in order to avoid overburdening any of them. When all SDN switches are linked to a single controller, this can result in a performance bottleneck. Distributed SDN controllers have been designed to address these fundamental difficulties. Implementing a distributed deployment of multiple controllers improves control plane scalability while also reducing the danger of single-point failures inherent in centralized installations (Dixi et al., 2014). Nonetheless, the distributed deployment technique may cause some controllers in the control plane to become overcrowded, while others remain underutilized (Y. Wang et al., 2017b). An SDN controller load balancing mechanism, which involves shifting switches across controllers, has been proposed to maximize the controller's forwarding capability within the control plane (Cello et al., 2017).

1.2 Motivation of the Study

The Traditional network architecture is characterized by its complexity, massive data flow, and different service expectations. SDN is a promising technology to address the open network challenges to meet network demands. However, the existing load balancing algorithms proposed using SDN do not focus on certain factors. First, the existing algorithms are lagging in considering aspects like prioritizing flows, featuring traffic dynamicity, failure recovery system, and agility.

As a result, it is critical to develop an effective load balancing algorithm for SDN that takes into account the aforementioned criteria in order to provide equal workload distribution among the available network resources. One of the key obstacles in deploying SDN is ensuring load balancing among numerous controllers. As network traffic and network size grow, a single centralized controller has difficulty spreading the load fairly across multiple controllers. As a result, multiple controllers were designed to handle these control plane difficulties. Current multi-controller deployments, on the other hand, are static, rendering the controllers incapable of responding to dynamic traffic patterns, resulting in load imbalances among them. (Sahoo et al., 2020). The introduction of programmability in SDN has spurred a growing interest among researchers in integrating machine-learning algorithms with SDN to enhance network performance.

As far as machine learning approaches are addressing a lot of issues in networking and other areas, we intend to address the switch migration based load balancing issues in distributed controllers through reinforcement learning algorithms which is a subdomain of ML, to make software defined networking successful and future network aspects.

1.3 Statement of the Problem

SDN provides a powerful programmable network framework for creating logical topologies and defining the appropriate location of network components. The separation of the data plane and control plane allows for efficient network logic administration. However, when all SDN switches are coupled to a single controller, a single SDN controller might become a performance bottleneck, particularly during periods of high incoming packet traffic (Foundation, 2012). Furthermore, the controller could be a single point of failure because it requires the suspension of all packet processing. Distributed SDN controllers are intended to address these two core issues (Belgaum et al., 2020). Each distributed SDN controller is in charge of its own network domain, with SDN switches statically assigned to specific controllers. As a result, this static allocation of switches to controllers might cause load imbalances among them, considerably leading to an unequal task distribution.

SDN employs multiple controllers to establish a scalable, highly accessible, and dependable network system. Nevertheless, the issue of load imbalance among these controllers remains a significant challenge. This load imbalance problem has a pronounced impact on the overall network performance. In recent years, the matter of load balancing across multiple controllers in SDN has garnered considerable attention as a captivating research area. When

incoming traffic places a heavy load on a controller in a multi-controller setup, it results in certain controllers becoming overloaded due to a lack of load balancing, while others remain underutilized.

In the event of load imbalance, SDN's multiple controllers may encounter the problem of controller load disparity. This issue arises from the distribution of switches to controllers through static configurations. Static mappings between switches and controllers prove inadequate in adapting to fluctuations in traffic loads, which can vary not only throughout the day but also across different locations within the network. Such variations occur over shorter time intervals (temporal) and in diverse network areas (spatial), ultimately resulting in load imbalance problems among controllers. Consequently, some controllers become overwhelmed while others remain underutilized.

In this approach, the overloaded controller is constantly identified as the controller with the highest load that exceeds the current load threshold. This solution, however, falls short of ensuring precise tracking of controller load as it approaches the processing capacity constraint. As a result, regular switch migrations are required, possibly disrupting ongoing traffic.

The proposed work focused on developing a reinforcement algorithm that can effectively achieve balanced load distribution for SDN multiple controllers based on a switch migration technique to improve performance.

1.4 Research Questions

This thesis aims to answer the following research questions:

RQ1: How to assess controller load imbalance and decide whether to conduct switch migration using a Reinforcement learning technique?
RQ2: How can we enhance current approaches to load balancing among controllers using a reinforcement learning algorithm?
RQ3: How to compare the performance of the newly proposed scheme against existing methods?

1.5 Objectives of the Study

1.5.1 General Objectives

The general objective of this research work is to design a switch migration based load balancing method for multiple controllers in SDN using Reinforcement learning approach.

1.5.2 Specific Objectives

In pursuit of our primary goal, the study will address the following specific objectives.

- To investigate the impact of controller load imbalance on network performance in SDN.
- To assess controller load imbalance and decide whether to conduct switch migration using a Reinforcement learning technique.
- To design a method that dynamically maps Controllers and switches to achieve balanced load distribution through switch migration using a Reinforcement learning algorithm
- To enhance current methods for switch migration based load balancing between multiple controllers using a Reinforcement learning algorithm.
- To evaluate the proposed method's performance with existing works.

1.6 Significance of the study

Machine learning-based balanced load distribution among several SDN controllers provides several significant benefits. It aids in load balancing, improving response times, throughput, and overall network performance. Furthermore, because of its centralized approach, it simplifies network device maintenance and facilitates configuration changes without the need for direct human participation.

Additionally, this approach can help organizations lower their capital expenditure by minimizing maintenance costs and reducing complexity. It fosters innovation through the use of standardized APIs, making it easier to develop and implement new solutions. Furthermore, it raises awareness among researchers interested in advancing load balancing studies.

1.7 Scope and Limitation of the Study

1.7.1 Scope of the study

Since time is the main constraint in this research, we have concretely focused on switch migration based load balancing and therefore, this research focuses on achieving balanced load distribution in distributed SDN controllers using machine learning-based switch migration. The proposed solution has been tested with different load scenarios and the discrete coefficient and discounted cumulative reward of the system have been displayed. Finally, the load balancing rate of the proposed system has been compared with RLB and SAR-LB that were raised in related works.

1.7.2 Limitations of the study

Several challenges have been identified in addressing load imbalance and the static connections between switches and controllers. However, it is important to note that this research will not encompass real network deployment, delve into controller security concerns, explore controller failure scenarios, or examine migration costs as far as our main concern is achieving balanced load distribution through multiple controller and would like to extend this concerns in the future work.

1.8 Organization of the Thesis

This thesis has been structured in the following ways:

Chapter One: - This chapter encompasses the comprehensive introduction, study motivation, problem statement, objectives, scope, limitations, and the research's significance.

Chapter Two: - The literature review section provides an in-depth exploration of prior research in the fields of machine learning and related works. It serves to enlighten the reader about the specific areas of focus within the thesis and the problems that have been addressed by previous studies.

Chapter Three: - In this chapter, we will delve into the methodologies and techniques employed throughout this research to develop the intended solution. Additionally, we will explore the evaluation metrics that have been utilized in our study.

Chapter Four: - This chapter states the overall proposed solution for the switch migration based load balancing using reinforcement learning.

Chapter Five: - In this chapter, we will provide the results of our experiments and participate in a thorough discussion of the findings.

Chapter Six: - This chapter will encompass a summary of our research findings, the formulation of conclusions, and the provision of recommendations for future research.

CHAPTER TWO

LITERATURE REVIEW

2.1 Overview of the Traditional Network

Traditional networking refers to the methods and protocols that had been established for linking computers and devices within a local area network (LAN) or a wide area network (WAN). It includes the cornerstone of networking technologies that have been in use for decades. Physical hardware devices such as routers, switches, and cables are essential in traditional networking for establishing connections and delivering data between different nodes. Ethernet and TCP/IP protocols manage data packet transfer, ensuring reliable communication across networks. Traditional networking often involves a centralized approach, where a single central device, such as a server, manages and controls the network activities (Kreutz et al., 2015)

Transport network protocols and distributed control mechanisms functioning within switches and routers are a critical technology in traditional networking topologies, enabling worldwide digital information transport. Despite widespread usage, traditional IP networks can pose administrative issues due to their intrinsic complexity. Network administrators must manually configure each network device using low-level and frequently vendor-specific instructions in order to articulate the desired high-level network policy. Because of the complexity of configuration, network infrastructures must constantly adjust to dynamic failures and load changes. Implementing basic rules in such a dynamic environment proves extremely difficult. In addition to the complexities of configuration, network environments must struggle with failure dynamics and adjust to changes in load, making the application of fundamental rules in this context particularly difficult (Benzekki et al., 2016).

The demand for services and network utilization is steadily increasing. Although growth drivers like video traffic, big data, and mobile usage boost profits, they also represent substantial problems for network operators. Spectrum congestion, the change to internet protocol (IP), and growing mobile customers are causing problems for mobile and telcom providers. Simultaneously, data-center administrators are dealing with a massive increase in the number of servers and virtual machines, which is boosting server-to-server communication traffic. To meet these difficulties, operators want an efficient, flexible, agile, and scalable network (Jammal et al., 2014).

Obtaining dynamic and autonomous network control without the need for physical intervention is one of the most difficult goals in network architecture. Nonetheless, it is one of the most valuable and unique objectives. Traditional networks lack automatic device configuration as well as the capacity to manage multiple activities. Network devices are deeply integrated into this system, with network traffic being governed by control elements and transmitted as digital packets through forwarding elements. In a conventional network, the data plane and the control plane are tightly coupled within networking devices, which can impede innovation, limit flexibility, and hinder the evolution of networking infrastructure (W. Lin & Zhang, 2016). In a standard network, these two planes are interconnected, resulting in a rigid and closed configuration., as it has been depicted in Figure 2.1 adapted from (Eissa et al., 2019)

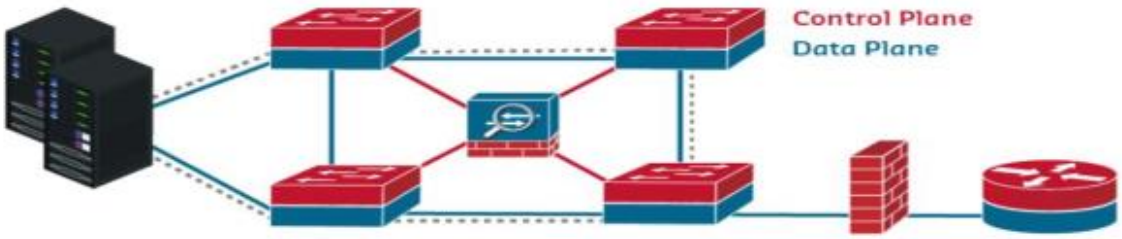

Figure 2.1 Architecture of Traditional Network

Traditional networking employs a centralized approach, in which a single central device, such as a server, is in charge of managing and directing network operations. The management plane develops new policies, which are then executed by the control plane. While it has provided a solid framework for network infrastructure, traditional networking is increasingly being supplemented or replaced by newer networking models that embrace virtualization, software-defined networking (SDN), and cloud-based solutions to meet the evolving demands of modern digital environments (Kreutz et al., 2015). SDN is a revolutionary network architectural paradigm that splits two planes to overcome the limits of traditional networks and provides several options for creativity. Figure 2.2 adapted from (Kreutz et al., 2015)shows three planes of SDN architecture

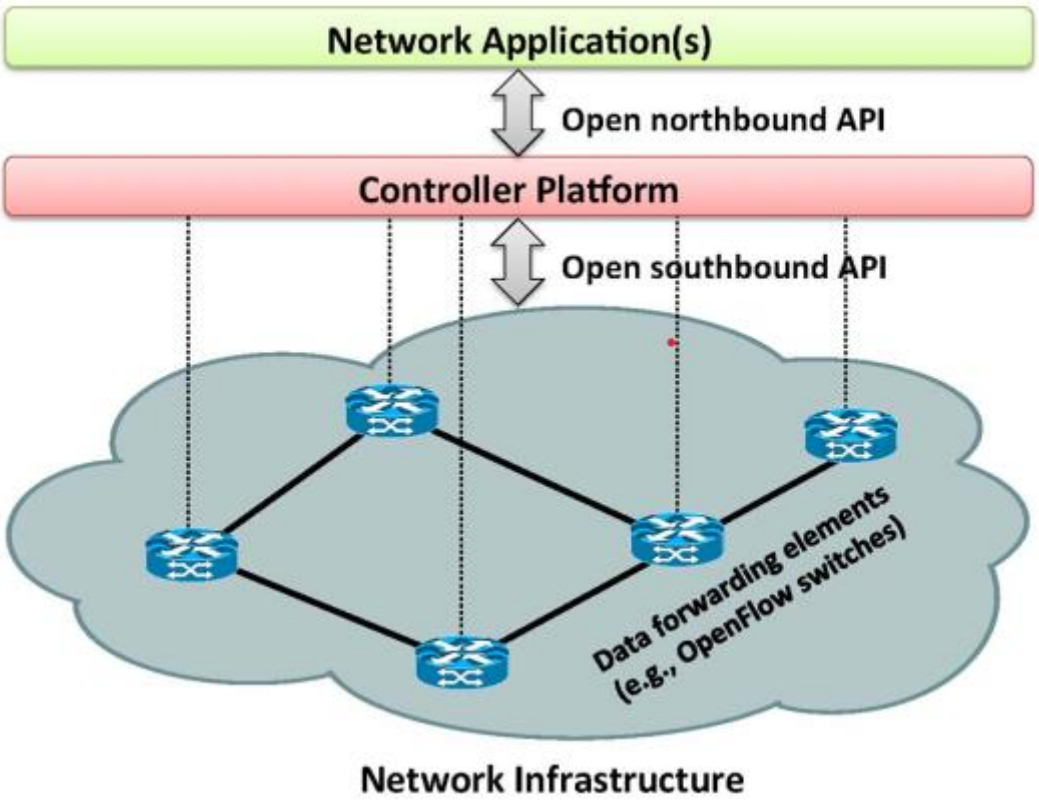

Figure 2.2 Simplified view of an SDN architecture.

SDN improves network performance in terms of management, control, and data handling. SDN is gaining recognition in areas such as cloud computing as a potential answer to the issues encountered by traditional networks. This technology is well suited for use in data centers and systems designed to handle certain workloads. Network managers can use Software-Defined Networking (SDN) to centrally manage data flow and change the behavior of network switches or routers. This is accomplished by implementing control applications as software modules, essentially reducing the need to manage each device independently (F. Hu et al., 2014).

2.2 Overview of SDN

SDN emerges as a possible future network technology solution, providing a modern and novel network architecture targeted at overcoming issues found in traditional IP-based networks. This novel network design allows for programmable network devices by separating the control plane from the data plane and achieving logical control centralization.

Because network layers are decoupled, an independent controller entity can adjust data routing laws within OpenFlow switches. This functionality allows SDN to quickly adapt to changing network requirements. Forwarding devices, which successfully interact with logically centralized controllers, supplement this separation. These controllers, in turn, supervise switch forwarding tables and are responsible for a variety of functions such as policy enforcement, topology management, connection discovery, network configuration, flow table entry management, communication flow packet forwarding, and topology management.

SDN is a sophisticated method of building, implementing, and managing networks that separate the network controller and forwarding devices to improve user experience. This network separation provides various benefits to the future Internet, including reduced complexity, controllability, flexibility, quick innovation, and better management (Hamdan et al., 2021; Verma et al., 2021).

SDN segregates the network into two distinct planes: the control plane and the data plane. Within this framework, switches are responsible for transferring actual data between hosts on the data plane, while the control plane governs the establishment of flow rules, which dictate how data is routed among switches. The separation of the data plane from the control plane serves to facilitate productive network logic administration (Han'guk T'ongsin Hakhoe. & Han'guk Chŏnja T'ongsin Yŏn'guso., 2012). If all SDN switches are coupled to the same controller, a single SDN controller can constitute a performance bottleneck in cases when incoming packets are significant. Furthermore, the controller is a potential single point of failure because it requires the suspension of all packet processing. To address these two major problems, distributed SDN controllers have been proposed(T. Hu et al., 2018). In a distributed SDN architecture, each controller is in charge of its own network domain, with SDN switches statically assigned to specific controllers. This static allocation, however, can cause imbalances in load distribution among controllers, resulting in uneven work distribution.

As a result, dynamic adjustments to the mapping of switches to controllers have been investigated in order to establish a more equitable workload distribution among distributed SDN controllers.

2.2.1 Architecture of SDN

As a new network paradigm, SDN represents a fundamental revolution in network technology, challenging established network foundational assumptions. The data plane, control plane, and application plane are the three separate layers of SDN. The data plane includes network equipment such as routers and switches that are responsible for forwarding packets depending on control plane directives. The control plane acts as a bridge between the application plane and the data plane, regulating network traffic flow management. The application plane, which is located atop the control plane, handles specialized application logic activities such as intrusion detection systems and large-scale data processing. To carry out its functions efficiently, it is frequently partitioned into many layers (Feamster et al., 2014), as depicted in Figure 2.3 adopted from (Amin et al., 2021)

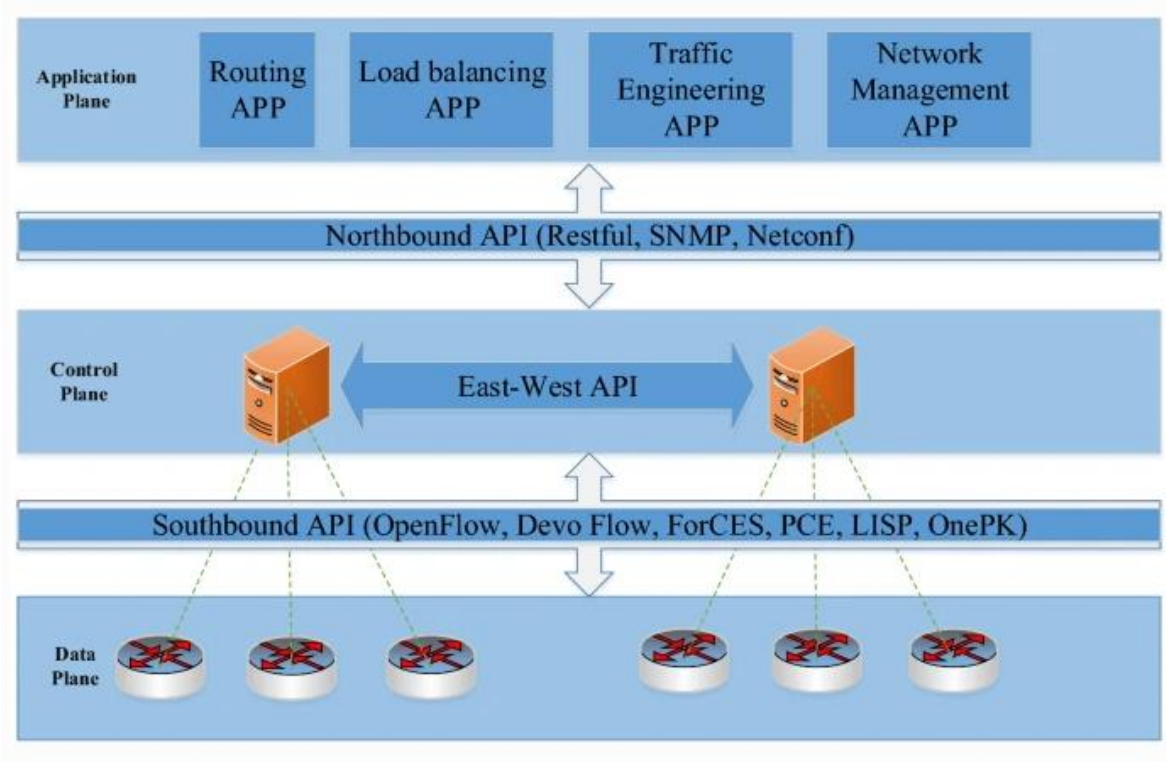

Figure 2.3 Detailed Layers of SDN Architecture

SDN, by definition, conceals the complexity of network design. Its architecture enables network control that is dynamic, cost-effective, controllable, and flexible.

Data Plane situated at the architecture's foundation, is alternatively referred to as the forwarding plane, user plane, or carrier plane. It comprises a set of network devices, whether virtual or physical, tasked with transmitting user traffic. The Data Plane operates by processing incoming frames based on the directives provided by the Control Plane. Actions

such as forwarding, modification, or discarding of frames can be executed within the Data Plane (Y. D. Lin et al., 2015).

Control Plane: The Control Plane is the primary center of network intelligence, performing critical functions like as routing and traffic signaling (Y. D. Lin et al., 2015). While it is designed to operate independently of the Data Plane, there are instances in which certain Control Plane components may be entrusted to network devices, resulting in a hybrid approach. The Southbound Interface (SBI), which was initially based on the OpenFlow standard, facilitates interaction between these two planes (Amin et al., 2018).

Application Plane: The Northbound Interface (NBI) connects the Application Plane, frequently via asynchronous means such as REST API, allowing the network administrator to precisely define the desired network behavior. Some authors choose to combine the Application and Control planes, while others keep them separate. The Control Plane is distinguished by the fact that it generally includes fundamental networking features that are universally shared among various applications, such as topology discovery and shortest-path calculation. The Application Plane, on the other hand, consists of individual programs that run on top of the Control Plane. SDN controllers are software platforms that unite the Control and Application planes in the context of SDN.

The major objective of the Management Plane is to enable network management for extra functionalities such as configuration, monitoring, billing, and so on. It covers a wide range of problems and is distinguished by its diversity. SNMP, HTTP, XML, RMON, and SSH are some of the protocols commonly used in this plane (Rojas et al., 2018).

To summarize, the SDN paradigm has a key advantage in that it provides logically centralized network control. It enables users to access both virtual and physical aspects through a unified interface by utilizing virtualized control planes and forwarding planes. Furthermore, when compared to traditional networks, SDN provides administrators with the opportunity to centrally monitor all network elements, enhancing global perspective management.

Because of SDN's architecture, approaches such as Machine Learning (ML) are gaining importance. These techniques have a wide range of applications, including resource QoS prediction, management, security, traffic engineering, and routing optimization (Amin et al., 2021).

2.3 SDN Controllers Overview

2.3.1 SDN Controller Evolution

In this section, let us start by outlining the history of multi-controllers with two instances. Following that, we will demonstrate the two basic multi-controller architectures: flat and hierarchical.

2.3.1.1 Single Controller

When SDN architecture where presented at the initial phase, they tend to use a single controller that controllers the entire network. As depicted In figure 2.4, adapted from (T. Hu et al., 2018) The controller (c1) in this network architecture supervises the operation of four switches. When a source host sends a new packet to switch (s1), the switch is caught off guard since it lacks the necessary routing data to carry out the forwarding task for this packet. As a result, the switch (s1) sends a "Packet-in" signal to the controller (c1) to request routing instructions for the incoming packet. The switch can then transmit the packet to the next device in the network when the controller gets the response message and gives the required guidance. Finally, the packet is transmitted successfully to the destination host. The controller, without a doubt, plays an important part in the traffic transmission process. However, its capacity is limited, and the increasing volume of flow requests generated by switches as network traffic grows rapidly can soon overwhelm a single controller. Furthermore, if the solitary controller fails, the switches are unable to identify the routing for fresh arriving packets, affecting network communication and applications. As a result, a new controller design has been proposed to overcome these issues.

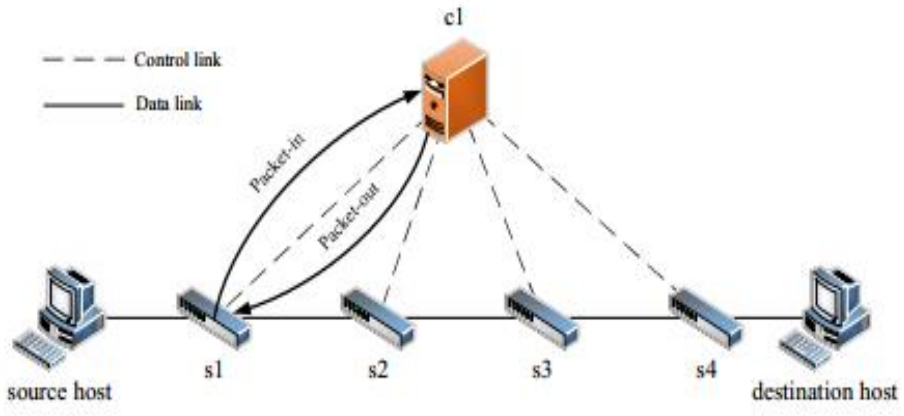

Figure 2.4 Single Controller

2.3.1.2 Multiple Controller

As OpenFlow protocol evolves, the concept of multi-controller setups arises as a novel SDN design approach capable of overcoming the limitations of a single-controller architecture. The network structure in Fig. 2.5 which is adapted from(T. Hu et al., 2018) shows two controllers, In this scenario, each of these controllers, (c1) and (c2), is in charge of a specific network section, they are conceptually centralized and follow the same logic. As a result, when new packets arrive at (s1), both (c1) and (c2) can quickly build forwarding paths across all related switches. This method effectively reduces the flow-processing load on a single controller. Furthermore, these two controllers act as mutual backups, potentially addressing the single point of failure vulnerability inherent in a single controller configuration.

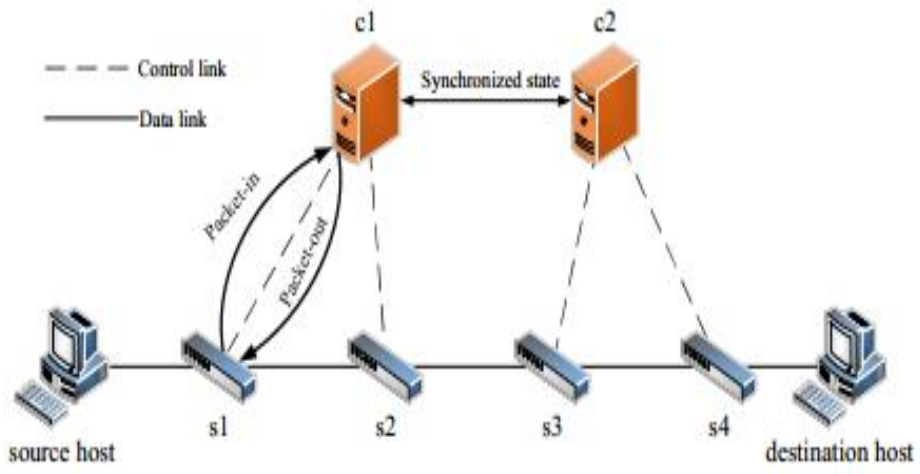

Figure 2.5 Multiple Controller

Based on the facts reported above, it is clear that a multi-controller design provides significant improvements in SDN network performance when compared to a single controller setup. As a result, research and interest in multi-controller systems have increased significantly in recent years.

2.4 The Basic Multi-Controller Architectures

When implementing a multi-controller system, the architectural design of these controllers is crucial. Following a thorough analysis of the existing literature, we determined that the fundamental multi-controller architectures may be divided into two categories: flat design and hierarchical design.

2.4.1 Flat Architecture

In the flat design approach, the network architecture is structured into distinct domains, where an individual controller controls each domain. These controllers have a deep understanding of their respective domains and possess localized network views, meaning they have knowledge and control over the specific devices, resources, and activities within their assigned domain.

To ensure coordination and collaboration between controllers, they establish communication through east-westbound interfaces. These interfaces enable the exchange of information, data, and commands between controllers, allowing them to synchronize their understanding and collectively build a comprehensive overview of the entire network.

By leveraging these east-westbound interfaces, the controllers can gather information from neighboring domains, acquire a broader understanding of the network's state, and make informed decisions based on this global view. This comprehensive overview enhances their ability to optimize network performance, allocate resources efficiently, handle network events, and ensure seamless operation across the entire network.

The flat design approach offers a decentralized and scalable architecture, allowing each controller to focus on a specific domain without being burdened by the complexity of the entire network. This distributed control model promotes modularity, flexibility, and adaptability, making it suitable for large-scale networks with diverse requirements and dynamic environments. Fig. 3 depicts the flat design of the multi-controller architecture. Typical examples are HyperFlow (Dotan & Pinter, 2005) and Onix (Shtykh & Suzuki, 2014).

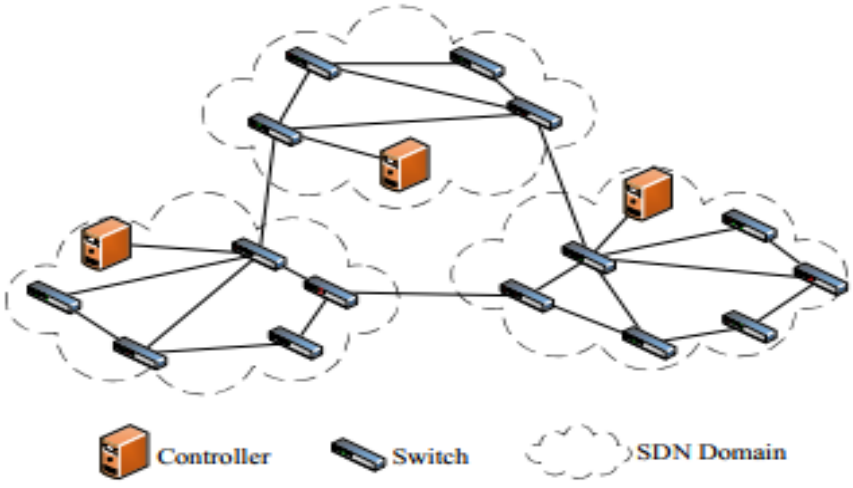

Figure 2.6 Design of flat multi-controller architecture

The flat design approach improves control plane capabilities, but it also introduces complications relating to controller management and increased control overhead. Controllers in this system must communicate often to maintain a consistent network view, which might be difficult. The hierarchical design method has emerged as a viable answer to these challenges.

2.4.2 Hierarchical Architecture

Hierarchical design is a widely employed approach in network management that involves the use of two-layer controllers. This design provides a structured framework for organizing and managing complex networks effectively.

The first layer in this hierarchical structure is the domain controller. Its primary responsibility is to oversee and control the switches within its designated local domains. A local domain typically refers to a specific section or segment of the network. The domain controller is equipped with the necessary intelligence and control applications to handle the unique requirements and operations within its assigned domains. By operating at this level, the domain controller can efficiently manage the switches and ensure optimal performance within its local domains.

The root controller is located in the second layer of the hierarchical design. This controller is crucial in supervising the entire network. Its primary task is to manage the domain controllers and maintain a complete view of the entire network. The root controller acts as a central hub for gathering data from domain controllers and making high-level network design, policy, and optimization decisions. It provides a global perspective that enables efficient network-wide management.

One prominent example of a hierarchical controller structure is Kandoo(Hassas Yeganeh & Ganjali, 2012). In the Kandoo architecture, the root controller serves as the central point of communication and coordination. It interacts with the domain controllers to acquire domain-specific information. The root controller relies on this information to gain insight into the state, performance, and configuration of each domain. By maintaining constant communication with the domain controllers, the root controller can effectively oversee and manage the entire network.

It is worth noting that in the hierarchical design, the domain controllers do not establish direct communication with each other. Instead, they solely communicate with the root

controller. This arrangement simplifies the network architecture and reduces complexity. The domain controllers focus on managing their respective local domains, while the root controller handles the coordination and consolidation of information from the domain controllers. This separation of responsibilities and communication channels enables efficient and scalable network management in hierarchical designs. Figure 2.7 adapted from(T. Hu et al., 2018) shows the basic architecture of hierarchical design.

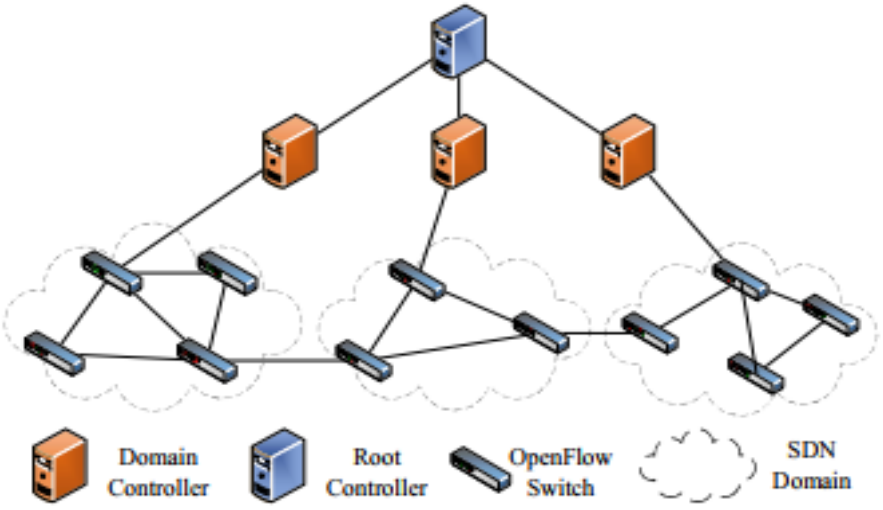

Figure 2.7 Architectural view of Hierarchical multi-controller design

While the multi-controller architecture efficiently addressed the problems associated with a single controller, it also introduced new ones. When multiple controllers are deployed, the network is divided into discrete SDN domains, with each controller in charge of the local switches inside its domain. Despite this, the network is vulnerable to controller overloading and underloading due to changes in network traffic and the static mapping of switches to controllers. Furthermore, unequal task distribution among controllers can dramatically affect network performance, resulting in issues such as high packet loss rates, delayed controller replies, and lower controller throughput.

2.4.3 Common Challenges in Multi-Controller Architecture

As the network grows in size, a single centralized controller cannot keep up with the increasing demand for flow processing. As a result, deploying multiple controllers appears to be the most practicable strategy to overcome the limits of a single controller. This, however, poses other issues, such as consistency, availability, security, load balancing, controller placement, and scheduling, which are prevalent in multi-controller settings.

Multi-Controller Scalability: The idea behind deploying a multi-controller system is to solve the constraints of a single controller, such as the danger of single controller failure and limited controller capacity. This is accomplished by utilizing two primary multi-controller concepts. Multi-controllers, on the other hand, present scalability issues, particularly in the selection of controller sites and the assignment of switches to different controllers within the network. Indeed, the scalability of a multi-controller system is affected by factors such as the number of controllers and the deployment style selected. If controllers are deployed arbitrarily, it may result in an imbalanced processing burden on controllers and a reduction in control plane capacity. Following our investigation, we divide existing solutions into two categories: (1) controller positioning; and (2) domain partitioning. Controller placement prioritizes picking optimal sites to promote scalability, whereas domain partition prioritizes separating the entire network into many SDN domains.

Multi-Controller Consistency: Using multiple controllers allows the entire network to be divided into domains, with each controller overseeing its own SDN domain. Controllers must engage in inter-domain information sharing to maintain a coherent perspective and enable the flawless transmission of packets across the network. As a result, in the context of multi-controller SDN, establishing controller consistency arises as a critical difficulty.

It is critical in the field of multi-controller systems that choices are made based on network information that is consistent and coherent. Nonetheless, concerns such as out-of-sync controller states and contemporaneous strategic conflicts across controllers may cause discrepancies in the controller's operational state during data transmission. Our review of previous research on multi-controller consistency can be divided into two categories:

(1) Controller State Consistency: This element is concerned with maintaining a consistent perspective inside specific local domains, especially when network conditions change.

(2) Control Strategy Consistency: This refers to the prevention of conflicts caused by controllers' flow table directives.

Multi-Controller Reliability: While using multiple controllers overcomes the issue of a single point of failure for controllers, it does not guarantee high control plane dependability. The connections between switches and controllers have a limited capacity. Controllers and switches may lose the capacity to communicate efficiently if these linkages become overcrowded, interrupted, or break, resulting in controller and switch isolation.

Furthermore, there is concern that malicious attacks, such as excessive packet-in requests, may disrupt or overload controllers. As a result, assuring the dependability of a multi-controller system is just as important for practical implementation. Current research on multi-controller reliability can be divided into two categories:

(1) Control Path Reliability: This viewpoint assesses the multi-controller design's capacity to establish reliable network connections.

(2) Controller Node Reliability: This aspect solves the issues of the multi-controller design by assuring trustworthy and dependable network nodes.

Multi-Controller Load Balancing: The use of multiple controllers divides the network into discrete SDN domains, with each controller supervising the local switches in its domain. However, due to the fluctuating nature of network traffic and the static mapping of switches to controllers, the network may encounter scenarios in which certain controllers are overloaded while others are underloaded. Furthermore, unequal load distribution among controllers will severely impact network performance (e.g., high packet loss rate, high controller reaction time, and poor controller throughput). Therefore, for a given multi-controller SDN network, it is essential to ensure better load balancing performance of the multi-controller.

Multi-controller load balancing should be investigated now more than ever due to several compelling reasons. As technology continues to advance and the demand for scalable and highly available systems grows, traditional load balancing techniques may no longer be sufficient to handle the increasing workload and complexity of modern applications. Therefore, we are dedicated to investigating the load balancing effectiveness in our concern as far as investigating and adopting multi-controller load balancing is essential in modern times to achieve scalability, high availability, performance optimization, adaptability, and enhanced security in complex and demanding application environments. By leveraging the benefits of multi-controller load balancing, organizations can build robust and resilient systems capable of meeting the evolving needs of today's digital landscape.

2.5 Multi-Controller Load Balancing

In a multi-controller SDN system, the network is divided into numerous SDN domains, each with its own controller. This segmentation allows for localized switch management inside

each domain. However, this design creates unique issues that have an impact on the network's overall performance. The dynamic nature of network traffic, as well as the static assignment of switches to controllers, provide significant challenges. Because network traffic is inherently volatile and can fluctuate over time, uneven workloads among controllers emerge. Furthermore, inflexible mapping of switches to controllers might result in certain controllers being overburdened with switches while others remain underutilized. This imbalance in workload distribution might lead overloaded controllers to struggle with the high volume of traffic, while underloaded controllers sit idle.

The consequences of such uneven load distribution are severe. Network performance is severely harmed, resulting in higher latency, increased packet loss, and a significant decrease in overall efficiency. These issues may eventually hamper the network's ability to deliver services dependably and effectively. To solve these problems and provide optimal performance in a multi-controller SDN network, strong multi-controller load balancing techniques need to be implemented. Load balancing involves distributing the workload evenly across the available controllers based on factors such as traffic patterns, switch utilization, and controller capacities. By dynamically reallocating the workload among controllers, load balancing mitigates the problem of overloaded and underloaded controllers.

Achieving good multi-controller load balancing performance requires a combination of intelligent algorithms, network monitoring, and dynamic reconfiguration capabilities. These mechanisms continuously monitor the traffic and performance metrics of the network, identifying areas where load imbalances exist. Based on this information, the workload is then redistributed across the controllers in a manner that optimizes overall network performance.

By ensuring balanced controller workloads, multi-controller load balancing promotes efficient resource utilization, minimizes congestion, and enhances the network's ability to handle varying traffic conditions. Ultimately, this leads to improved network performance, increased reliability, and better user experiences. As a result, for a given multi-controller SDN network, it is critical to ensure good multi-controller load balancing performance. Our analysis of the literature led us to the conclusion that our current research solves the problem in two ways: (1) controller clustering and (2) switch migration. Controller clustering is primarily concerned with architectural design by establishing a dynamic controller resource

pool, whereas switch migration is concerned with dispersing controller loads in order to accomplish and maintain load balancing.

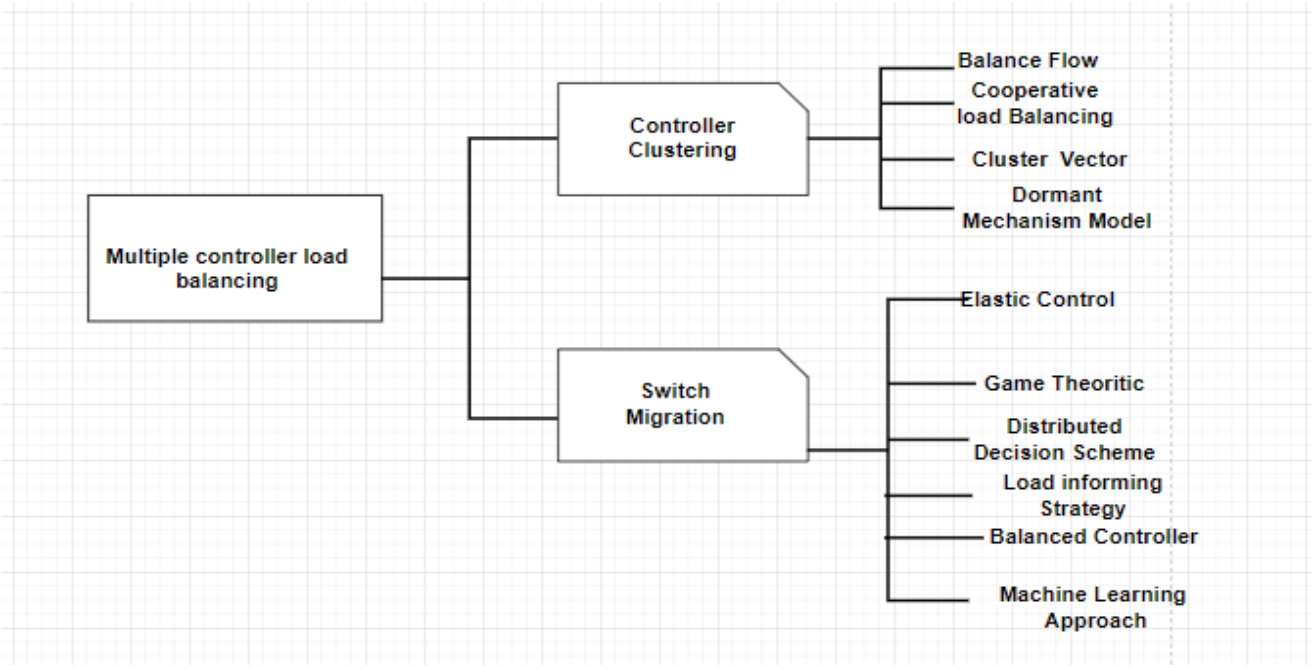

Figure 2.8 Multiple controller load balancing classification:

2.5.1 Controller Clustering Method

The controller clustering strategy divides the network into a supercontroller and multiple standard controllers, which form the controller resources pool. The supercontroller is solely responsible for handling all controller workloads and collecting flow statistics from ordinary controllers in each domain regularly. Each regular controller carefully monitors its assigned domain and routinely provides load data via a cross-controller interaction mechanism. When traffic loads are high, the supercontroller starts a load balancing mechanism and assigns specific switches to various controllers. The supercontroller manages load balancing efficiently without imposing additional unneeded burden on the normal controllers by centralizing and collecting load data via controller clustering (Y. Hu et al., 2013; Selvi et al., 2016; Sufiev & Haddad, 2017).

1. **BalanceFlow:** BalanceFlow is a popular controller clustering method that uses a hierarchical deployment strategy. The ability to fine-tune the flow requests processed by each controller without introducing additional propagation delays is a key advantage of this technique. It is similar to the multi-controller capabilities found in OpenFlow 1.2. Every controller in the BalanceFlow architecture stores its load information and exchanges it with the others via regular communication via a cross-

controller mechanism. When traffic patterns change, one of the BalanceFlow controllers becomes the supercontroller. In response to changing traffic patterns, this supercontroller is responsible for traffic segmentation and reassigning various flow configurations to the appropriate controllers. BalanceFlow also suggests an acceptable switch extension action: the CONTROLLER X action. By implementing this extension action, the overloaded controller can reduce the processing stress of flow requests by dynamically reallocating such requests to controllers with a lesser workload (Y. Hu et al., 2013).

2. **Cooperative Load Balancing:** Similar to BalanceFlow, cooperative load balancing introduces a supercontroller to oversee the load management of individual controllers. In contrast, the Cooperative Load Balancing Scheme for Hierarchical Controller Deployment (COLBAS) is developed, which provides an alternate strategy based on cooperative efforts among controllers achieved through cross-controller communication. The basic idea behind this method is similar to that of BalanceFlow but with a different approach. The authors of COLBAS use a Greedy technique to rebalance workloads among controllers. Notably, COLBAS seeks to maintain excellent system performance while minimizing the cost of load reassignment. (Selvi et al., 2016).
3. **Cluster Vector (CV):** Load balancing is carried out using a self-defined CV label, which is expressed as a vector containing the addresses of controllers in the same cluster. This method avoids the necessity for the supercontroller and the normal controllers to be interdependent. The proposed concept was split into two levels: The supercontroller manages high-level operations in this approach, whereas normal controllers oversee low-level operations. Each controller keeps its own CV, and a regular controller uses it to find the addresses of other regular controllers and communicate with them about their respective workloads. (Sufiev & Haddad, 2017).
4. **Dormant Mechanism Model:** Some researchers have created a dormant mechanism model for multiple controllers based on a flat design to conserve network resources, reduce power consumption, and optimize controller use. When network demand is low, the basic idea is to put idle controllers into a dormant state, where they are inactive or powered down. (Fu et al., 2014).

2.5.2 Switch Migration Method

Researchers established the notion of multi-controller load management via switch migration in the context of a dynamic multi-controller architecture. Certain switches are reassigned to a controller in another domain with a lesser workload when a controller becomes overloaded or encounters a significant rise in flow requests from switches within its domain. The fundamental idea underlying switch migration is to dynamically reallocate the relationships between switches and controllers by moving switches from overburdened controllers to those with a lighter demand.

1. **Elastic Control (ElatiCon):** ElastiCon is a cutting-edge switch migration framework based on a dynamic multi-controller architecture. This flexible approach is made up of four basic modules: load measurement, load adjustment, decision-making, and action modules. The load measuring module is in charge of monitoring each controller's load and communicating this information to the load adjustment and decision-making module. To accomplish dynamic control over both controllers and switches, the load adjustment and decision-making module analyses how the load should be distributed among the controllers. The action module then performs control actions, such as switching, adding, or removing controllers, to apply the estimated load distribution and accomplish dynamic control of the network's controllers and switches. ElastiCon analyses the load on each controller regularly, finds any imbalances, and redistributes the burden among the controllers autonomously by shifting switches from an overloaded controller to one with a reduced load. This automated method ensures that the network's controllers have a balanced load distribution (Dixi et al., 2014).
2. **Game Theoretic Model:** The switch migration strategy employs game theory in this case. A zero-sum game is played with underloaded controllers as game players and switches as commodities. The model is used to simulate the abilities required to move switches between congested controllers. Although this paradigm is quick and effective for switching, it is not ideal for big networks. Because of the method's great complexity (Chen et al., 2016).
3. **Distributed Decision Scheme:** This method recognizes the Switch Migration Problem (SMP), which is concerned with optimizing the relocation of switches among controllers in order to optimize resource utilization and network performance. It also handles the Network Utility Maximization (NUM) problem, which tries to

maximize network resource efficiency given specific restrictions. To address these issues, the Distributed Hopping Algorithm (DHA) was created to accomplish the most efficient switch migration possible in order to improve network resource utilization and accommodate a higher volume of service requests within the available control resources. (Cheng et al., 2015).

4. **Load Informing Strategy:** A distributed decision architecture was built in this technique, which includes four main components: load estimation, load reporting, load balancing decision, and switch migration. Each controller in this architecture routinely exchanges its load information with the other controllers. These controllers not only collect and retain load data from their peers, but they also perform periodic load informing. While periodic load informing can help to speed up decision-making, it does introduce extra processing and communication overhead to the control plane. This is especially noticeable when the current load level does not fluctuate significantly from the prior measurement, rendering the reporting procedure redundant across multiple controllers. (Y. Wang et al., 2017a).
5. **Balanced Controller (BalCon):** This heuristic technique is founded on two significant observations. To begin, considering the communication patterns within the SDN network is critical for efficient switch migration. Second, switches and switch migration should be managed centrally. It is especially advised to replace switches with strong connections, especially those with the shortest distance to the controller. BalCon significantly minimizes the frequency of migration modifications while working with the same controller, especially in small-scale networks. (Cello et al., 2017).
6. **Machine Learning Approach:** Network performance can be considerably improved by integrating Machine Learning (ML) techniques with the Software-Defined Networking (SDN) architecture. SDN enables network operators to centralize certain logic on the control plane, allowing for the acquisition of a comprehensive network picture suitable for the operation of ML algorithms. This combination of SDN with ML provides enhanced network management capabilities for optimization, automation, and adaptation.

As far as Machine learning is being explored for switch migration load balancing due to the complexity of network environments and dynamic traffic patterns. It enables optimization and resource allocation by considering factors like switch utilization and traffic demand.

Machine learning algorithms can make real-time decisions, adapt to changing conditions, and scale effectively. They offer self-learning capabilities to continually improve performance. While still a research area, machine learning holds promise for efficient resource utilization and enhanced network performance.

2.6 Machine Learning (ML) Algorithms in SDN Load Balancing

Machine Learning (ML) is a subset of Artificial Intelligence (AI) that enables systems to learn from experience and improve their performance without the need for explicit programming (Ethem, n.d.). It directs the development of systems. ML systems are capable of making judgments and identifying various patterns. These ML models handle new data on their own, relying on their capacity to make decisions, perform calculations, and produce outputs by learning from previous computational states. ML has given computers decision-making skills, boosting computing systems' intelligence. This newly discovered capability has potential uses in SDN, particularly in the control layer, where it can act as the SDN architecture's decision-maker (Shirmarz & Ghaffari, 2020). ML is a wide field with methods that have been divided into numerous groups. Techniques in machine learning are often classified based on the type of learning they include, with a noteworthy contrast between *supervised, unsupervised, and reinforcement learning.*

A. Supervised Learning

Supervised learning is a labeled learning strategy in which algorithms are given a labeled training dataset that contains inputs and their known outputs. As a result, the algorithms can build a system model that captures the learned link between inputs and outputs. Following training, when new input data is brought into the system, the trained model can be used to predict the predicted output.

1. **K-Nearest Neighbor (k-NN):** K-NN is a supervised learning technique that relies on the classification of a data sample, which is determined by taking into account the unclassified sample's k-nearest neighbors.
2. **Decision Tree (DT):** Decision Trees (DT) are a classification technique that works by generating a learning tree. Each node in this tree structure refers to a data feature (attribute), branches indicate feature combinations that result in classifications, and each leaf node represents a specific class label. To classify an unlabeled sample, compare its feature values to the decision tree nodes to establish the correct class assignment.

3. **Random Forest:** The random forest method, also known as a random decision forest, is a versatile methodology that could be used for classification and regression applications. It is made up of several decision trees. To solve the issue of overfitting associated with individual decision trees and improve accuracy, the random forest method adopts a strategy in which just a portion of the feature space is randomly selected to form each decision tree.
4. **Neural Network (NN):** A neural network is a computational model based on the human brain that is made up of interconnected processing units known as neurons. It learns from historical data and uses this information to create predictions or decisions. The network is divided into three levels: the input layer, which receives data, the hidden layers, which analyze and transform information, and the output layer, which provides final predictions or choices. Neural networks adapt and improve their performance by adjusting the strength of connections between neurons during training, allowing them to excel in tasks such as image recognition and natural language processing, making them an important tool in modern machine learning and artificial intelligence.
5. **Support Vector Machine (SVM*):*** Another widely adopted supervised learning technique for tasks involving classification and pattern recognition is the SVM. The fundamental concept behind SVM is to project input vectors into a higher-dimensional feature space. This transformation is accomplished through the utilization of various kernel functions, including but not limited to linear, polynomial, and RBF.
6. **Bayes' Theory:** The Bayes theorem uses conditional probability to calculate the likelihood of an event occurring, taking into account prior knowledge about conditions that could be related to the event.
7. **Hidden Markov Models (HMM):** Hidden Markov Models (HMMs) are Markov models. Markov models, which adhere to the memoryless feature, find widespread use in contexts characterized by random dynamics. The memoryless property of Markov models implies that the conditional probability distribution of future states is simply impacted by the current state's value and is independent of all preceding states.

B. Unsupervised Learning

At their foundation, unsupervised learning algorithms seek to discover patterns, structures, or insights inside unlabeled data by grouping comparable data samples. These techniques are widely used in tasks like clustering and data aggregation, where the primary goal is to group data points based on their intrinsic commonalities. Widely used unsupervised learning algorithms are-

1. **K-means:** The k-means algorithm is a popular unsupervised learning technique for partitioning an unlabeled dataset into discrete clusters. The K-means approach is commonly implemented with only two parameters: the initial dataset and the desired number of clusters.
2. **Self-Organizing Map (SOM):** SOM is a well-known model in unsupervised neural networks. SOM is commonly used for applications like data grouping and dimensionality reduction.

C. Reinforcement Learning

Reinforcement Learning is an algorithm for making decisions. It is a training based on the ML approach, that promotes good behaviors while penalizing undesirable ones. This ideal behavior is learned through interactions with the environment and observation of how it responds. It is similar to how children assess their surroundings and identify behaviors that help them achieve their goals.

The learning agent in RL is taught to assign positive values to useful acts and negative values to detrimental behaviors. As a result, there are two kinds of reinforcement: positive and negative. To get an optimal solution, the agent is motivated to seek the largest significant long-term total gain. Long-term goals also prevent the agent from becoming fixated on little objectives while learning through contact with the environment. All judgments in reinforcement learning are made sequentially. This means that the state's output is determined by the present input, and the next input is determined by the prior input. Because the learning in this case is dependent, we label sequences of dependent decisions.

Reinforcement Learning (RL) is another widely adopted learning technique applied in the context of Software-Defined Networking (SDN). RL comprises three key components: an agent, a state space denoted as S, and an action space denoted as A. The agent, acting as a learning entity, engages with its environment to acquire knowledge about the optimal

sequence of actions aimed at maximizing a long-term reward. This long-term reward is a cumulative measure that considers both present and future benefits, incorporating a discount factor to account for the time value of rewards.

The issue of Internet traffic modeling is still in the works. RL algorithm is a well-studied machine learning approach and has been used in several MDP-modeled situations. In RL, a learning agent detects the environment and acts at each state to achieve a well-defined objective and maximize the collected action rewards. According to its learning capabilities, the RL algorithm has the advantage of being able to make time-series judgments in a discrete space while updating action rules, which is critical for load balancing difficulties in SDN. Thus, the fundamental goal of this work is to employ reinforcement learning to learn traffic behavior in order to balance the network load in a dynamic environment. We employ an online ε-greedy based Q-learning technique for reward cost optimization as well as exploration and exploitation.

2.4 Switch Migration in Load Balancing of Distributed SDN Controllers

Multiple SDN controllers can control a switch in distributed SDN controllers. However, only one controller may manage the switch for PACKET IN requests and flow rule installation. The southbound interface protocol specifies the relationship between the switch and the controller.

Dynamic switch migration presents a promising strategy for achieving elastic scaling and load balancing in practical SDN environments. This approach has typically employed in three key scenarios:

1. Capacity Expansion: When the combined traffic load surpasses the capacity of all existing controllers, it becomes necessary to introduce new controllers into the network. During this process, switches have been redistributed to these new controllers to ensure efficient load management.

2. Controller Deactivation: In situations where a controller has either shut down or put into a low-power mode to conserve energy and reduce communication overhead, the switches previously managed by that controller need to be migrated to other active controllers.

3. Load Balancing: Even without changes in the total number of deployed controllers, switch migration becomes essential for load balancing purposes. This involves relocating selected switches from overloaded controllers to controllers with lighter workloads, ensuring that no

individual controller exceeds its operational capacity. This operation is commonly referred to as load balancing.

Additionally, to ensure efficient resource utilization, switches can be dynamically reassigned from an overloaded controller to one with lighter loads through role modifications between controllers. This dynamic load distribution strategy ensures even work distribution among distributed SDN controllers. Consequently, multiple controllers can collectively handle PACKET IN requests originating from switches (C. Wang et al., 2017).

2.5 Related Work

Load balancing research for SDN controllers has been ongoing for a while now. However, the traditional switch migration strategy frequently falls short of successfully managing the trade-off between load ratio deviation and migration needs. Furthermore, survey results show that a major amount of existing literature favors switch migration as the principal strategy for developing load balancing systems. Nonetheless, this reliance on switch migration introduces several issues, such as the use of static migration selection criteria, inefficiencies in the migration process, and the possibility of conflicts during migrations.

The work by (C. Wang et al., 2017) proposed a framework *Switch Migration-based Decision Making* (SMDM) where they compute *load diversity*, a ratio of controller loads. Their study focuses on enhancing migration efficiency through the use of SMDM in optimal migration efficiency scenarios. The greedy technique used to create the SMDM method consists of two phases: load balancing detection and migration action generation. When the load imbalance between two controllers surpasses a certain threshold, switch migration occurs. The switches chosen for migration have lighter loads and higher efficiency.

The work of (Ye et al., 2017) introduced a resource-intensive model for SDN as a first step in their approach. This approach accounts for a wide range of controller CPU, bandwidth, and memory requirements across several switches, with switches serving as resource consumers and controllers serving as resource suppliers. Then SMP is reduced to a network utility maximization problem, which is a typical combinational problem. Second, they introduced the Distributed Hopping method (DHA), a synthesizing distributed method for distributing computing work among participating controllers. Finally, they put their idea into action by developing DHA-CON, a proof-of-concept based on the Beacon controller. The effectiveness of their strategy was validated using several numerical simulations, which confirmed its performance.

In the work of (T. Hu et al., 2017), This study addresses the challenge of load balancing among multiple controllers in SDN. It introduces the Stated Switch Migration Strategy (SSMS) to achieve effective controller load balancing in multi-domain networks. SSMS comprises two stages: the first stage uses a genetic algorithm to select migration entities between overloaded and underloaded domains, and the second stage sequentially migrates switches with specified parameters. While SSMS demonstrates effective load balancing in distributed SDN networks, it may have limitations in large-scale networks and doesn't address switch and controller selection or dynamic traffic conditions with heavy loads.

In the work of (Jingmei Li;Xinliang Hu;Mengqi Zhang, 2018) To solve the issue of load imbalance across SDN multi-controllers, they introduced FMOPSO-DSMS, a dynamic switch migration method based on fuzzy multi-objective particle swarm optimization. This technique seeks to optimize several critical objectives, including load balancing, average minimum transmission hop count, and switch migration frequency. It successfully balances controller load while lowering network transmission delay and switch migration expenses. However, it is worth mentioning that the algorithm's computing and data storage requirements were significant, potentially causing network instability.

In the work of (Begam et al., 2022a) They developed a technique known as Multiple Regression-Based Searching (MRBS) to improve server selection and routing path performance in Data Centre Networks (DCN), even in demanding conditions such as message spikes, changing message frequencies, and unexpected traffic patterns. MRBS uses regression analysis to anticipate traffic patterns and response times based on server metrics such as load, response time, bandwidth, and utilization. MRBS combines a heuristic algorithm with a regression model to optimize server and path selection.

In the work of (Sun et al., 2020) They presented MARVEL, a dynamic strategy for balancing controller workloads that uses multi-agent reinforcement learning to produce switch migration operations. MARVEL is divided into two phases: offline training and online decision-making. During the training phase, each agent actively engages with the network to learn the process of migrating switches. Following that, during the online phase, MARVEL used to make educated switch migration decisions.

In the work of (Zhou et al., 2018) They introduced a dynamic approach for load balancing via switch migration by changing the logical architecture of SDN. The primary goal was to improve the scalability and security of the control plane. Their novel method involves

modeling switch migration as a signature-matching problem, which is then represented as a three-dimensional Earth Mover's Distance (EMD) model. They also prioritized durability and computational efficiency, which resulted in the creation of suboptimal algorithms with polynomial time complexities. The simulation results showed that these algorithms efficiently minimize traffic discrepancies across controllers and protect them from reconnaissance and saturation attacks.

The work by (T. Hu et al., 2019) developed a method called Efficiency-Aware Switch Migration (EASM) to evaluate load balancing by calculating the normalized load variance across controller loads. The goal of EASM is to improve controller load equilibrium and migration efficiency. Its primary job is to migrate switches and achieve load balance among controllers while taking into account parameters like performance, reaction time, load balancing rate, and migration efficiency. The results show that EASM effectively reduces controller reaction times, improves controller performance, reduces migration costs, and increases load balancing rates. However, it does not account for packet loss and may not perform efficiently during periods of high incoming traffic loads. Additionally, EASM relies on a threshold based on the difference between the maximum and minimum controller loads to identify imbalances in controller loads when the load difference matrix exceeds this threshold.

The work by (Filali et al., 2019) This research finds into distributed SDN control plane designs, with a focus on controller load balancing. The study proposes a system for scheduling switch migrations to increase migration efficiency and ensure balanced controller loads. The researcher used the ARIMA time series model to anticipate switch load. When an overload has been predicted, the algorithm plans a migration operation to relieve the situation. The accuracy of the ARIMA forecasting model has been evaluated, as well as the algorithm's performance, by measuring controller reaction times. The report recognizes the ongoing difficulty of static mapping between switches and controllers. Furthermore, the authors use a specified threshold to detect overloaded controllers.

The work by (Jiru et al., 2023) introduced the Improved Switch Migration Method (ISMM), which is aimed at balancing controller workloads by shifting switches from overloaded controllers to those with lighter loads. ISMM is critical in assessing the state of controllers during the load balancing process. It detects heavily loaded switches within overloaded controllers and migrates them to the most appropriate controllers, optimizing the utilization

of clustered resources. ISMM deployment resulted in considerable improvements in a variety of performance parameters, such as throughput, response time, and packet loss. ISMM, on the other hand, relied on predefined threshold values, resulting in inferior migration outcomes and failing to account for load fluctuations in diverse network conditions.

The work by (Yeo et al., 2021) proposed an approach known as switch-aware reinforcement learning load balancing (SAR-LB) in Distributed SDN Controllers for achieving balanced Load Distribution with Reinforcement Learning-Based Switch Migration. They balance load distribution between SDN controllers by using dynamic Switch migration. This work employs a reinforcement learning, controller, and switch selection system known as SAR-LB for switch migration. As neural network inputs, SAR-LB leverages the utilization ratio of numerous resource categories in both switches and controllers. Switches are also treated as RL agents to reduce learning activity while considering all migration possibilities. The experiment results reveal that SAR-LB delivers nearly even load distribution amongst SDN controllers due to precise decision-making during switch migration. However, their approach does not show the effect of the switch migration in different load scenarios.

The work by (Kidist, 2022) proposed new Open Flow Model based Multi-Controller Topology. When compared to a single controller topology using random load balancing algorithms in SDN POX controller, the proposed topology can reduce response time (sec) by an average of 30.12%, increase transaction rate (trans/sec) by an average of 39.44%, and increase throughput (KB/sec) by an average of 10.56%. however the proposed methodology does not deal the ideal mechanisms that should be used to measure controllers workload. And also it is focused on traditional method of load judgments and used the traditional algorithm which can not guarantee effective load distribution.

Table 2.1: Related Works Summary

Title	Methods/Algorithms	Work Focus/Key concepts	Limitation
SDN Controller Load Balancing Based on Reinforcement Learning (Li et al., 2019)	• RL	• Designed a selection algorithm in the phase of switch migration as one switch migration triplet. • Made the use of reinforcement learning for switch migration load balancing under multiple controllers.	• The method was not set methods to select an appropriate switch and the target controller for migration. • Tested only under the same network condition. • Tested on small-scale network
Achieving Balanced Load Distribution with Reinforcement Learning-Based Switch Migration in Distributed SDN Controllers(Yeo et al., 2021)	• DQN	• Introduced the SAR-LB scheme, which leverages switch-aware reinforcement learning for load balancing. • Utilized resource utilization ratios from both controllers and switches as the neural network inputs. • When compared to other competing methods, the method improves the normalized standard deviation of controllers by up to 34%.	• Since the method utilizes a neural network, it faces scalability challenges due to the conversion of feature vectors into a fixed-size image format. • Sets static resource utilization threshold values which results in performance inaccuracy.

Load Balancing in DCN Servers through SDN Machine Learning Algorithm(Begam et al., 2022a)	• MRBS	• A heuristic estimator is used to ensure that routing is free of congestion, assisting in the best path to the server. • Introduced MRBS algorithm, which improves throughput by adding a heuristic path technique and applying a heuristic estimation function. • Improved the efficient use of network resources and data pertaining to servers, including aspects such as load, response time, and bandwidth.	• Does not deal with a large network. • The work does not deal with dynamic controller and switch migration.
AI-Assisted Framework for Green-Routing and Load Balancing in Hybrid Software-Defined Networking(Etengu et al., 2020)	• RNN-LSTM • ARIMA, FARIMA, BPNN. • DNN • Q-learning • DDPG	• Provide an overview of the most recent optimization techniques for global energy-efficient routing and load balancing. • Investigate a scalable and intelligent integrated architectural framework that uses deep reinforcement learning (DRL) approaches.	• Missing synthetic and real experiments and comparisons. • Does not deal with distributed controllers and switch migration schemes. • It focused on a hybrid network rather than pure SDN.
Deep Reinforcement Learning- based load balancing strategy for multiple controllers	• DQN • MDP	• SDN modeling analysis is performed using the Markov Decision Process (MDP) to yield the system state, migration action set, and system reward.	• Linear optimization was used, which is not well suited to the practical workload of SDN controllers, causing difficulties in

in SDN(Xiang et al., 2022)		• A Deep Reinforcement Learning (DRL) architecture is intended to address the Switch Migration Problem (SMP) for multiple controllers.	recognizing currently overloaded resource categories. • And also results in significant time overhead.
MARVEL: enabling controller load balancing in Software-Defined Networks with multi-agent reinforcement learning(Sun et al., 2020)	• DRL + RNN	• Model the distributed control plane as a multi-agent system to tackle the SMP in a distributed fashion. • Design a Deep Reinforcement Learning (DRL) framework for each agent in the MARL model.	• MARVEL models the load balancing of individual controllers using linear optimization. • It has a limited amount of switch migrations in its action space.
ESMLB: Efficient Switch migration load balancing for Multicontroller SDN in IOT(Sahoo et al., 2020)	• ESMLB	• Introduced a framework that employs preliminary iterative calculations for each switch-controller pair before making decisions.	• Computational overhead can impact both the response time and scalability of the load balancing process.

2.6 Summary of Related Works

In summary, numerous techniques for multiple controller load balancing based on SDN have been proposed in recent years. However, the proposed solution still has certain flaws. To address the issue of controller load imbalance for multiple controllers in SDN, two solutions have been used majorly so far: controller clustering and switch migration are the common techniques that have been used as main techniques to solve the load imbalance issues. Later on, after the popularity of machine learning, different machine learning techniques came into action to solve the load imbalance problem. Even though machine learning techniques are employed in SDN load balancing techniques, they can't achieve optimal switch migration load balancing due to different load measurement techniques employed in the system.

System balance can be disrupted by fluctuating network traffic, reducing performance. Understanding these traffic patterns is critical for making educated decisions. This study uses reinforcement learning to provide a framework that understands network traffic complexities and selects lightly loaded controllers for efficient load balancing. The framework learns to predict traffic fluctuations by analyzing past data, maintaining stability, and improving overall system operation.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Overview

This chapter outlines the methodologies, tools, and procedures employed for analyzing the research challenges. It offers a concise overview of the techniques applied in designing, implementing, and evaluating the proposed solution. Furthermore, it provides an in-depth exploration of the overall methods, data collection processes, and evaluation procedures. Crucially, this section offers an academic rationale for the chosen approach, aligning with the research's primary objectives

3.2 Methodology

The following strategies and techniques are used to attain the specified aims and answer the research questions:

Stage 1: Literature Review

comprehensive review of numerous sources, including research papers, periodicals, conference papers, guideline manuals, and journal articles, is undertaken to achieve the research objectives. The goal of this review is to provide a complete overview of the research domain and important work on the subject.

Stage 2: Problem Formulation

following a thorough review of various recent articles and papers related to the switch migration based load balancing, the research problems were initially formulated. Subsequently, these formulated problems underwent multiple rounds of review and refinement.

Stage 3: Design Proposed Solution

The new method was proposed by leveraging switch migration, a technique employed to adjust controller load distribution. This adjustment is achieved by transferring switches from overloaded controllers to underloaded ones using reinforcement learning. Once a switch migrates from its initial controller to the target controller, the target controller takes responsibility for receiving and processing all loads generated by that switch.

Stage 4: Extract the data from Arnes network topology

We employ Python-based simulations to assess the performance of the proposed strategies. We have used the Arnes network topology from the Internet Topology Zoo. Along with the topology, we employ synthetic data generated through Python code that matches our assumptions about the influx and servicing of flows in switches.

Stage 5: Simulating the proposed system.

During this phase, we will perform a simulation using Jupyter Notebook. This simulation will involve the integration of a reinforcement learning algorithm designed for load balancing within a distributed controller framework and the data generated from the Arnes network topology, coupled with a switch migration scheme.

Stage 6: Reporting, evaluating, and discussing the result.

To assess the effectiveness of our proposed solution, this phase will involve a comparative analysis with existing load balancing mechanisms to evaluate its performance. We will gauge performance-using metrics such as average discrete coefficients, cumulative reward, and load ratio. The research outcomes, derived from simulation results, are elaborated upon in this phase. Visual aids, including charts and diagrams, simulation results. MS Office Word will serve as the primary writing tool for this thesis.

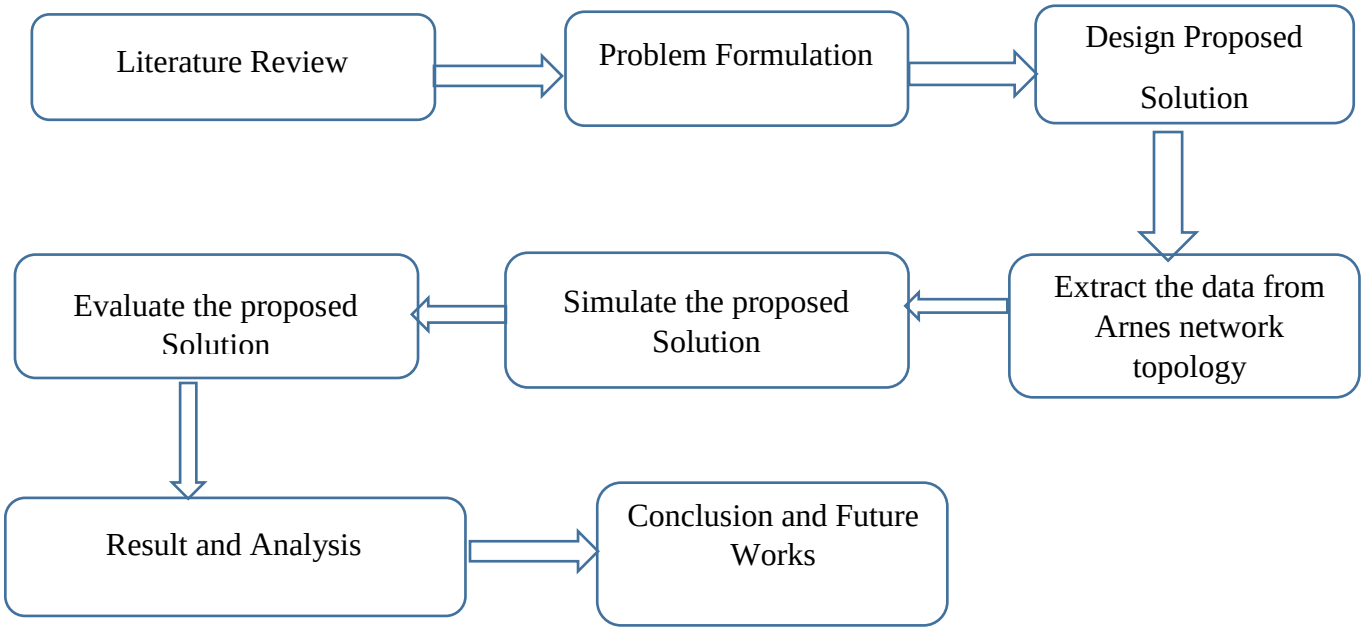

Figure 3. 1: Proposed methodology of the Study

3.3 Data Collection Method

To evaluate the performance of the proposed schemes, we use Python-based simulations. We use the Arnes network topology from the Internet Topology Zoo[1]. Along with the topology, we make use of synthetic data generated from python code, we have generated 6801 from the network topology for each load scenario and we get 20,403 data, which fits with our assumptions regarding the influx and servicing of flows in switches.

Using the network topology that we have selected to meet the requirement of the SDN network, we have used the following codes to build a graph for the simulation where the network data is from Arnes.gml.

A sample code for data collection, cleaning, and handling corner cases.

```
    return table_data

    def build_graph(k, graph, final_data):
    lat = nx.get_node_attributes(graph, "Latitude")
    lon = nx.get_node_attributes(graph, "Longitude")

    # Remove bad nodes from 'graph'
    for node in list(graph.nodes):
        # removing nodes
        # (a) with undefined latitude or longitude
        # (b) that are isolated
        check_bad_node = lat.get(node) == None or lon.get(
            node) == None or nx.is_isolate(graph, node)
        if check_bad_node == True:
            graph.remove_node(node)
```

Figure 3. 2 A sample code for graph data cleaning

A code for data table used for Q-learning generated from the graph

[1] http://www.topology-zoo.org/index.html

```
import numpy as np
def generate_table(arrival_lambda_list, service_mu_list, switch_count):
    print("Arrival Lambda = {}".format(arrival_lambda_list))
    print("Service Mu = {}".format(service_mu_list))

    # Generate the data
    data = []
    event_count = 100

    for i in range(0, switch_count):
        arrival_times = np.random.exponential(
            1/arrival_lambda_list[i], event_count)
        arrival_timestamps = np.cumsum(arrival_times)
        service_times = np.random.exponential(
            1/service_mu_list[i], event_count)
        departure_timestamps = arrival_timestamps + service_times

        for j in range(0, event_count):
            data.append((arrival_timestamps[j], i, 1))
            data.append((departure_timestamps[j], i, -1))
```

```
    # Sorts the data as per the first element (timestamps in our case)
     data.sort()
     table_data = []

     temp_list = [0] * switch_count

     for i in range(0, len(data)):
         timestamp = data[i][0]
         switch_number = data[i][1]
         update = data[i][2]

         temp_list[switch_number] += update
         table_data.append([item for item in temp_list])

     return table_data
```

Figure 3. 3 A sample code for data generation from graph

3.4 Research tools

3.4.1 gym

The gym is a Python tool that works within an open-source framework. It makes it easier to create and compare reinforcement learning algorithms by offering a consistent API for coordinating learning algorithms with different contexts. It also provides a standard set of environments that comply with this API. Gym's API has become generally recognized as the industry standard for performing such operations since its release. (Hock et al., 2013)

The Gym library enables researchers to create custom-built environments and provides the Q-learning agent as a useful resource for this research. OpenAI Gym is a platform for academics to design and test learning agents. It was designed specifically for reinforcement learning agents and is highly suited for such tasks.

3.4.2 Hardware Tools and Software Tools

Table 3. 1 Hardware and Software tools used

Tool	Type	Description
Python	Programming Language	Python is an object-oriented, high-level, dynamically semantic programming language that is interpreted. It is popular in data science, artificial intelligence, and machine learning.
Jupyter Notebook (ubuntu)	software	An interactive, web-based computing notebook environment in which you can modify and execute simple documents while narrating the data analysis process.
Microsoft Word	software	Used to write and prepare documents
Mendeley Desktop	software	It is most known for its reference management tool, which is used to organize and disseminate research papers as well as to generate bibliographies for academic articles.
Draw.io	Online tool	An online tool for drawing diagrams, architectures, UML, and other types of drawings
Scikit-learn	Python Library	This Python package includes a variety of strong statistical modeling and machinelearning capabilities.
NumPy	Python Library	The Python library NumPy is used for array manipulation and handling.
TensorFlow	Python Library	This Python module is intended for doing quick numerical computations. It is a fundamental framework that can be used to build Deep Learning models directly or through wrapper libraries developed on top of TensorFlow, simplifying the modeling process.

Networkx	Python Library	Networkx is a Python module that allows you to create, manipulate, and examine complicated network structures, as well as their dynamics and associated functions (Hagberg et al., 2008).
Matplotlib	Python Library	Matplotlib, a Python toolkit, provides a comprehensive set of tools for creating static, animated, and interactive visualizations.

3.5 Evaluation Metrics

Discrete Coefficient of Controllers: The discrete coefficient between the controllers is defined as the degree of deviation of load between the controllers. A smaller discrete coefficient means better results(Li et al., 2019).

$$D_{(C_i,C_j)} = \left(\sqrt{\sum_{k=i,j} \left({R'}_{C_k} - \overline{R_{C_i,C_J}} \right)^2 /2} \right) / \overline{R_{C_i,C_J}} \qquad \textit{Equation 3. 1}$$

Cumulative discounted Reward: The discounted reward is taken over all time points as a running sum. This reward should converge to a particular value in order for the Q-Learning to be considered valid(Li et al., 2019).

$$r_{discounted}(n) = r(s,a) * \gamma^n \qquad \textit{Equation 3. 2}$$

Cumulative Discrete Coefficient: The discrete coefficient of all controllers is defined as the degree of load balance of the complete control plane, which will be one of the indications used to evaluate the selection algorithm's performance (Li et al., 2019). Hence, in this research, we have used cumulative discrete coefficients as a performance evaluation metric.

$$D_{(C_i,C_j)} = \left(\sqrt{\sum_{i=1}^{n} \left(R_{C_k} - \bar{R} \right)^2 /2} \right) / \bar{R} \qquad \textit{Equation 3. 3}$$

CHAPTER FOUR

PROPOSED SWITCH MIGRATION BASED LOAD BALANCING

4.1 Introduction

In this chapter, we delve into the proposed solution aimed at addressing the issue of controller load-imbalance when multiple controllers are in use. The chapter encompasses several key aspects, including an overview of the proposed solution, a detailed explanation of the general design, an exploration of the dynamic controller discrete coefficient, and an examination of the underlying assumptions. Additionally, we discuss critical topics such as switch selection, the selection process for target controllers, and the dynamic adjustment of discrete coefficients for the controllers.

The SDN with many controllers is the context of our research. There are three controllers in Figure 3.1, and each controller is linked to many switches to form a subdomain. When switches attached to controller C2 generate a significant number of flow requests, C2 may become overloaded and other controllers' resources may be underutilized. To meet load requirements, switches handled by the overloaded controller can be transferred to other controllers, according to OpenFlow1.3. As indicated in Figure 3.1, V4 is chosen to be migrated, and C1 is chosen as the target controller. After the migration, V4 is handled by C1, and the complete network is operational. As a result, switch migration can alleviate the controller load imbalance problem (Xiang et al., 2022). The main challenge is how to migrate efficiently, so a switch migration strategy needs to be designed to efficiently implement controller load balancing.

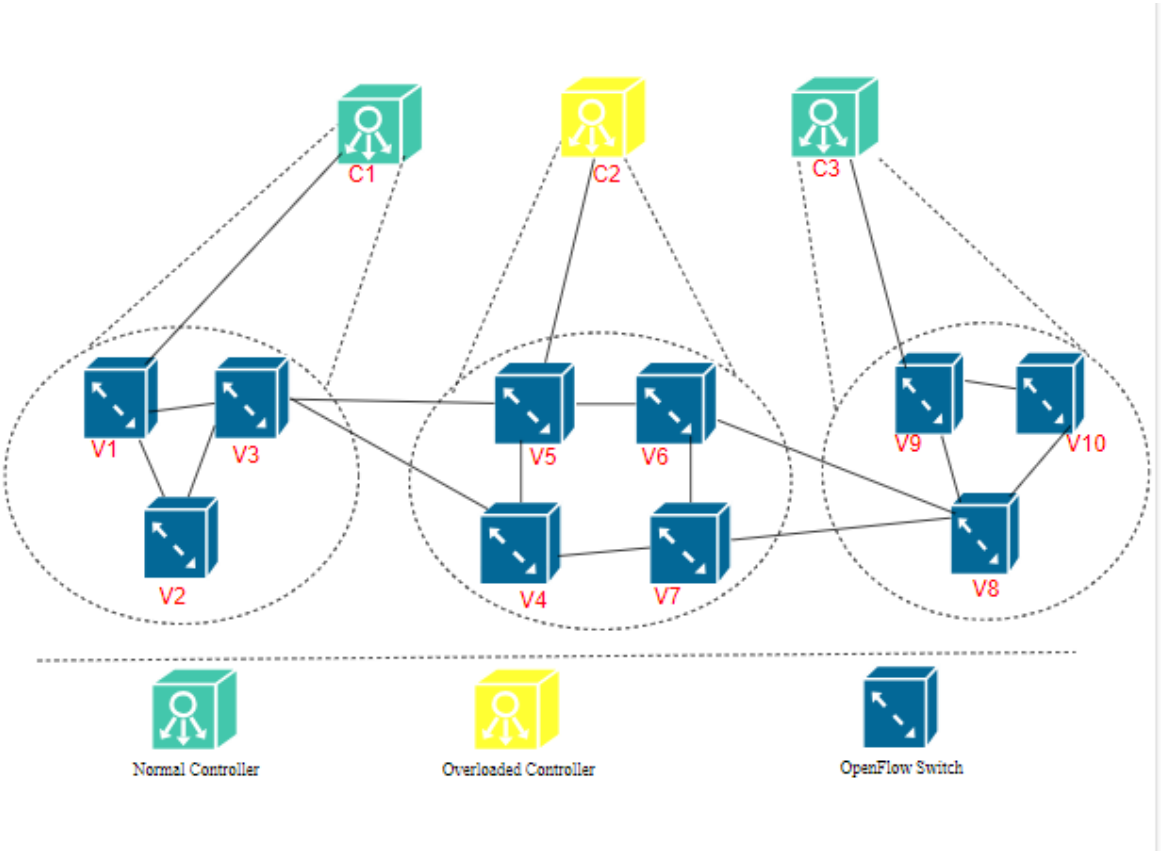

Figure 4. 1 Multiple Controller Switch Migration Scheme

4.2. Proposed Solution

4.2.1 Overview of the proposed Switch migration Method

Switch migration is the process of moving a switch's functionality from one controller to another. This migration often entails transferring switches from an overloaded controller to an underloaded controller, with the primary goal of distributing load more evenly among the controllers. This procedure allows for dynamic changes to the controller load, resulting in more balanced network management. However, existing switch migration designs have encountered difficulties in obtaining ideal load-balancing performance. Varieties of investigations have revealed the dynamic and spatial differences in network traffic situations. As a result, certain controllers may become overloaded while others stay underloaded at various times and locations. When the load on a controller exceeds its predefined maximum threshold, a process is activated to shift the switches currently managed by the overloaded controller to those that are underloaded. This proactive method can't guarantee that network resources are used efficiently and controllers operate within their capacity limits (H. Wang et al., 2018). Indeed, a switch can establish connections with multiple controllers within the network, playing roles such as a slave or having an equal

partnership with other controllers. Additionally, one of these controllers may assume the role of the master controller, coordinating the operations of the switch within the network domain. (T. Hu et al., 2019)(Beiruti & Ganjali, n.d.).

Figure 4.2 depicts the switch migration procedure between two controllers, namely Controllers 1 and 2. Switch 3 is being migrated from Controller 1 to Controller 2 in this migration scenario in order to distribute the workload more evenly between these two controllers.

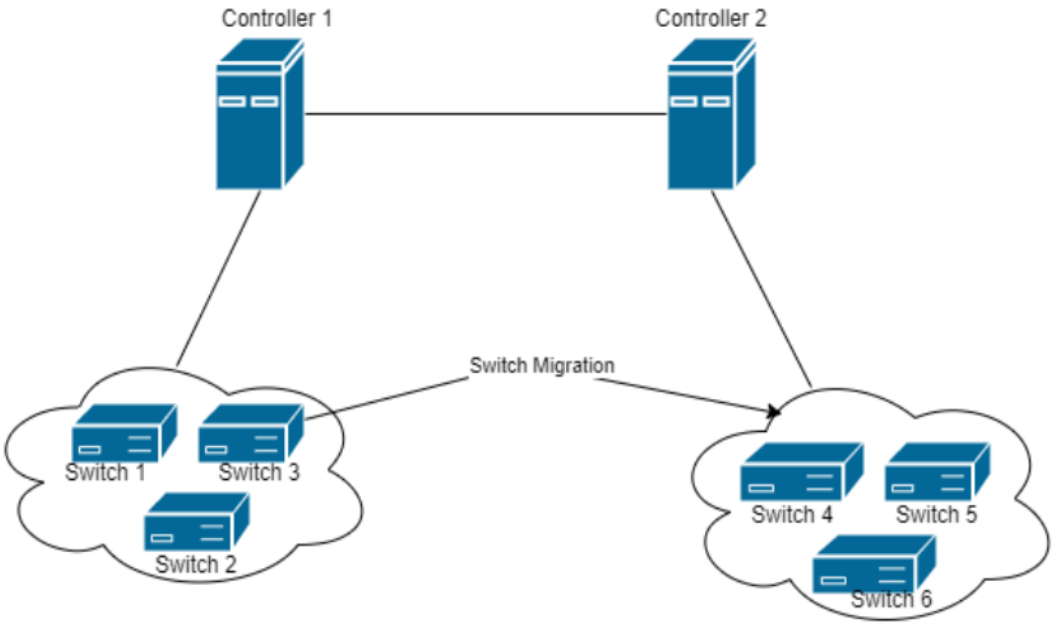

Figure 4. 2 Overview of Switch Migration Method

The switch migration method is a highly successful strategy for addressing load imbalance concerns in a network with multiple controllers. The suggested system is intended to distribute workload equitably between various controllers, supporting efficient load balancing and overall network performance enhancement. Failure to adopt such procedures can lead to a decrease in system performance.

4.2.2 Overall Design of Proposed Solution

In this section, we discuss the switch migration design. Researchers have discussed that by appropriate switch migration, the overload trend among the controllers can be reduced, thus enabling the existing controllers to manage their resources effectively to handle the requests from the switches. This section introduces several terminologies: controller's load and load ratios, migration efficiency, and discrete coefficients of controllers. The selection algorithms are designed to generate migration tuples, consisting of the out-migration domain, the in-

migration domain, and a set of migration switches, which will also be used in the reinforcement learning process discussed in the subsequent chapters. Some important symbols, that are used in this chapter are described in Table 4.1.

Table 4. 1 List of symbols for parameters in load balancing

Symbol	Description
LCi	The controller Ci's load measurement.
CCi	The metric for evaluating the load capacity of controller Ci.
RCi	The metric for assessing the load ratio of controller Ci
LSk	The rate at which switch Sk sends packet-in messages.
R	The average load ratio measure across all controllers.
D	The measure of load ratio deviation coefficient between two controllers.
E	The measure of switch migration efficiency between controllers.
HSk	The metric indicating the number of hops between the controller and the switch.

4.2.2.1 Load Balancing

The first task at hand balancing the controllers' load. In this section, we discuss 3 parameters, namely, load, load ratio, and discrete coefficient.

A. Controller's Load (L*Ci*)

The load of the SDN controller is primarily burdened by the following factors(Y. Wang et al., 2017b).

- When the switch cannot discover the necessary forwarding flow entry, it sends a packet-in message to the pre-assigned controller.
- In the control plane, controllers communicate with one another.
- The controller configures the flow entries for the switch.
- The network view of the complete control platform is maintained and shared by all controllers.

The quantity of packet-in messages received by the controller is the key factor contributing to controller load. The controller load is simplified in this study by aggregating the rates of packet-in messages processed by the controller. Which can be defined as follows-

$$L_{C_i} = \sum_{k=1}^{n} L_{S_k}$$ *Equation 4. 1*

B. Controller's Load Ratio (L_{Ci})

Controllers exhibit variable load capacities influenced by factors like CPU performance, the number of processors, and memory size, among others. The load ratio of a controller is defined as the proportion of its load to its load capacity. Consequently, the mean load ratio of the controllers represents the average of these load ratios across all controllers.

$$\begin{cases} R_{C_i} = L_{C_i}/C_{C_i} \\ \bar{R} = \sum_{i=1}^{n} R_{C_i}/n \end{cases}$$ *Equation 4. 2*

C. Discrete Coefficient (D)

The discrete coefficient for all controllers is a measure of the overall load balance within the entire control plane. It serves as one of the indicators employed to assess the performance of the selection algorithm. A lower value indicates more effective load balancing within the control plane.

The discrete coefficient between two controllers can be defined as follows-

$$D_{(C_i,C_j)} = \left(\sqrt{\sum_{k=i,j} \left(R'_{C_k} - \overline{R_{C_i,C_j}} \right)^2 /2} \right) / \overline{R_{C_i,C_j}}$$ *Equation 4. 3*

When we consider multiple controllers the discrete coefficient becomes –

$$D_{(C_i,C_j)} = \left(\sqrt{\sum_{i=1}^{n} \left(R_{C_k} - \bar{R} \right)^2 /2} \right) / \bar{R}$$ *Equation 4. 4*

4.2.2.2 Switch Migration Efficiency

Migration overhead is introduced during the process of moving a switch between controllers. This overhead is caused by differences in the hop count (HSk) between the switch and the controller, as well as the load (LSk) associated with the switch and the controller. Migration efficiency is quantified as follows.

$$E_{(C_i,S_k)} = L_{S_k}/H_{S_k} \qquad \textit{Equation 4. 5}$$

4.3 System Model

In this section, we will first outline the load balancing problem before formulating it using the Markov Decision Process (MDP). In this scenario, we will look at four parameters: state, action, reward, and cost.

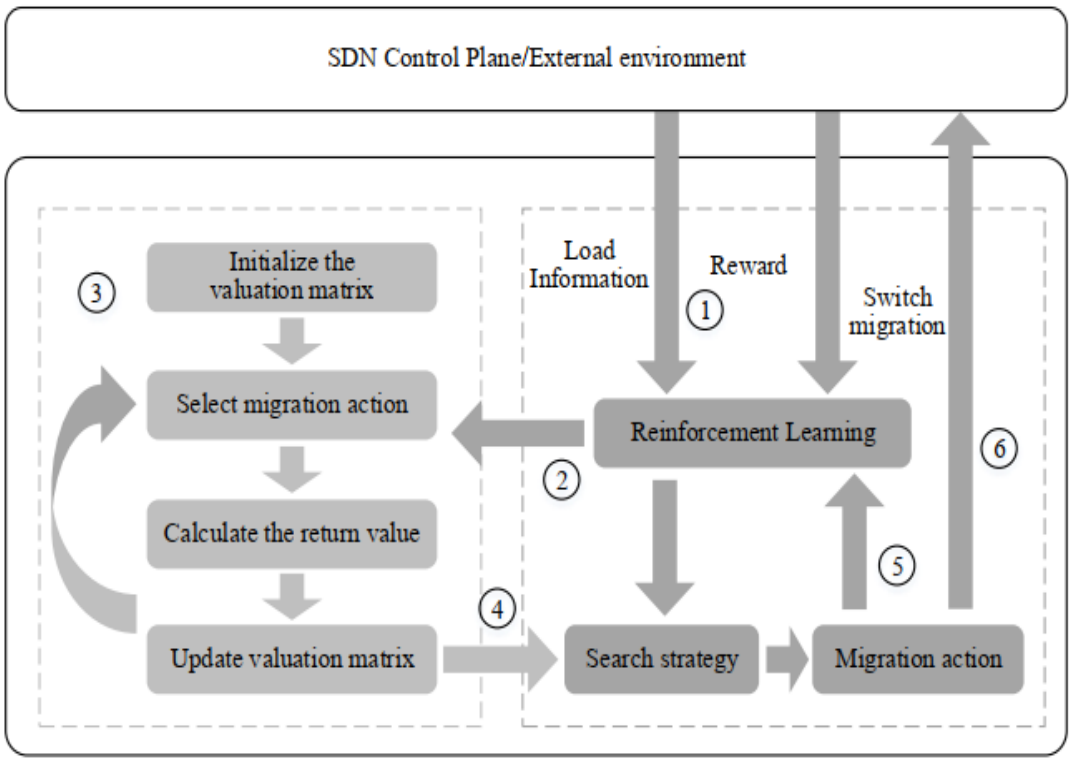

Figure 4. 3 Reinforcement Learning-Based Optimal Migration Model.

4.3.1 General Assumptions

We consider a system where a switch is assigned to or controlled by a controller at a time *t*. Thus,

$$x_{ij}(t) = \begin{cases} 1 \; if \; a \; link \; exists \; between \\ switch \; and \; controller \\ 0 \; , otherwise \end{cases} \qquad \textit{Equation 4. 6}$$

Since we are in a multi-controller setup, a controller can be mapped to zero or more switches. Every switch has a limited capability of processing flows. We assume that the maximum number of flows supported by each switch is N_F.

The number of new flow arrivals at the switch s_i in time *t* is denoted by $s_i(t)$. A controller's load is defined as the total number of flows that reached it at time *t*. The instantaneous load of a controller C_j at time *t* can be computed as below.

$$L_{C_j} = \sum_{i=1}^{n} s_i(t) x_{ij}(t) \qquad \textit{Equation 4. 7}$$

In this study, we consider an ideal load of a controller as the mean load of the system i.e.,

$$L^I(t) = \frac{\sum_{j=1}^{k} L_{C_k}(t)}{K} \qquad \textit{Equation 4. 8}$$

We want to minimize the difference between the ideal load which is calculated by equation 4.8 and the instantaneous load which is computed by equation 4.7. For this purpose, we introduce the discrete coefficient given by equation 4.4 and use it as a decision factor to determine the quality of our implementation.

Table 4. 2 List of symbols for describing problem statements in Q-learning

Symbol	Description
S	A collection of switches., $\mid S \mid = n$
C	A collection of controllers, $\mid C \mid = k$
$s(t)$	the rate at which switch s requests flows at a specific time t.
$L_c(t)$	The real-time load experienced by controller c at a specific time t.
$L_I(t)$	Ideal load experienced by all the controllers collectively at a specific time t.

4.3.2 State and Action

We assume that for each of the switches s_i, the flow arrivals can be modeled with a Poisson process with parameter λ_i. The service times are considered to be exponentially distributed with means $\frac{1}{\mu_i}$. The arrival of a new flow or the departure of a serviced request is taken as a decision epoch. Since the system at hand has a **Markovian** nature, we observe the system

state at each of the decision epochs. We do not need to consider the system state at any other time points except for the decision epoch because the system does not change elsewhere.

4.3.2.1 State Space

In our problem, the state space *S* is a set of tuples consisting of the number of flows in each controller. Each state $ST \in S$ corresponds to a tuple described by {number of flows in *C*1, number of flows in *C*2, ..., number of flows in *Ck*}.

4.3.2.2 Action Space (A)

At every decision epoch, an action chosen at the current state moves the system to the next state. Since at every decision epoch only one controller *Ci*, either gets or loses a flow, we define the action $a \in A$ for that controller. Action space for C_i is {do nothing, add a switch to C_i, remove a switch from C_i}.

4.3.2.3 Transitions and Transition Probability

As the system moves from state *s* to a state s', we can define a transition probability for the same. From each state-action pair (*s*,*a*), the system moves to a different state s' with finite probability $P_{SS'}(a)$. We will later consider Q-Learning which is a model-free algorithm, but for the sake of understanding let's consider an example.

Let's say we have 2 controllers and 3 switches. From our assumption, we have N_F as the maximum number of flows a switch can support. Let's consider that $N_F = 2$. Under the action *do nothing* we have –

$$P_{SS'} = \frac{\lambda_1}{\sum_{i=1}^{3} \lambda_i + 6\mu} \qquad \textit{Equation 4. 9}$$

4.3.3 Reward and Costs

When a system is at the state *s*, and some action *a* is chosen, the reward has to be some fairness metric which has to be decided upon in a way that reflects the long-term outcome of the action. Similarly, for every action we take, we incur some overhead which we have to define as a cost. For every switch migration between the controllers, we have to make sure that the cost is minimized. We define these terms as follows:

4.3.3.1 Reward (R)

The reward is taken to be negative of the discrete coefficient, i.e.,

$$r(s, a) = -D_{ij} \qquad \textit{Equation 4. 10}$$

This is reasonable since the discrete coefficient is a representation of the degree of disorder in the system of controller loads. If we take an action *a* that increases the disturbance, then the reward must be more negative. On the other hand, an action that stabilizes the system must receive a smaller negative reward.

4.3.3.2 Cost (C)

If no action is taken then no cost should be incurred. On the other hand, for every switch exchange, flows are redirected to the new controller, which is the reason for an overhead. We define the cost function as follows –

$$C(s,a) = \begin{cases} 0, if\ the\ action\ is\ do\ nothing \\ number\ of\ flows\ exchanged\ between\ controllers, \\ otherwise \end{cases} \quad \textit{Equation 4. 11}$$

4.4 Proposed Solution Design

In this section, we propose three ways for a load-balancing mechanism. The first one is by selecting the switch and target controller randomly, and the second is a completely greedy approach that works by generating tuples for migration. The third one works by using an on-line Q-learning algorithm and makes use of a Q-table.

4.4.1 Random Approach

For each time step, we calculate the ideal load and then classify the controllers in I_{domain} and O_{domain}. One of the overloaded controllers $C_i \in O_{domain}$ is selected and undergoes load balancing by migrating a switch S_j randomly from $C_i.s$ to the target controller which is selected randomly. This process continues until it gets balanced.

4.4.2 Greedy Approach

For a pure greedy implementation, we proceed in two steps. First, we divide the controller sets into two domains, i.e., out and in-migration domains. Since the controller load changes fast in a real-time changing environment, collisions in the control plane are common. The mechanism we use for migration domain selection ensures that collisions are avoided by assessing each controller appropriately. This decision is made considering the average load rate and discrete coefficients presented in the previous sections.

Algorithm 1: Selection Algorithm of Migration Domain

input: $R = \{R_{C1}, R_{C2}, ..., R_{Cn}\}$: Controller's load ratio

$\bar{R}$: Mean load ratio of all controllers

D: Discrete Coefficient between the controllers

output: Out-migration domain O, in-migration domain I, migration queue Q

initialize: Set O, I, and Q to empty sets

begin

foreach $R_{Ci}, R_{Cj} \in R=\{R_{C1}, R_{C2}, ..., R_{Cn}\}$ **do**

  $D_{Ci,Cj}$: Calculate the discrete coefficients

  if $D_{Ci,Cj} > D$ *and* $R_{Ci} > R$ *and* $R_{Ci} > R_{Cj}$ **then**

    Add Ci to the out-migration domain O

    Add Cj to the in-migration domain I

  if $D_{Ci,Cj} < D$ *and* $R_{Ci} < R$ *and* $R_{Ci} < R_{Cj}$ **then**

    Add Cj to the out-migration domain O

    Add Ci to the in-migration domain I

  Add (Ci,Cj) to the migration queue Q

Now that we have our domains separated, we focus on optimally selecting switches to migrate between them. For this, we leverage switch migration efficiency and load rate reduction as two decisive factors for selecting switches. The ideal switch to migrate should reduce the maximum load ratio under the best migration efficiency. Also, each switch migration should reduce the discrete coefficient between the controllers.

Algorithm 2: Selection Algorithm of Migrating Switch

input: Migration Queue Q

output: Switch collection S', migration tuples T

initialize: Set T to an empty set, n = 0

begin

foreach $q = \{Ci,Cj\} \in Q$ **do**

  foreach $Sk \in Ci$ **do**

    Calculate E_{Sk}

    $E_{Ci,S} \leftarrow -E_{Ci,S} + E_{Ci,Sk}$

    $L \leftarrow -E_{Ci,Sk}$

6 sort L

7 $\overline{E_{Ci}, S} \leftarrow - E_{Ci,S}/n$, $k = 1$

8 **while** $E_{Ci,Sk} \in L$ *and* $S_k \in S$ **do**

9 Calculate R'_{Ci}, R'_{Cj} *and* $D'_{Ci,Cj}$

10 **If** $E_{Ci,Sk} \geq \overline{E_{Ci,S}}$ and $D'_{Ci,Cj} <$ temp **then**

11 K++ ,temp $\leftarrow -$ $D'_{Ci,Cj}$, *add* S_k *to* S'

12 **else**

13 break

14 **T** $\leftarrow -$ S'

4.4.3 Q-learning with ε-greedy

In this section, we are going to take a look at the proposed Q-learning algorithm and understand the theoretical aspects of the same before taking a look at the step-by-step implementation of the same.

4.4.3.1 Proposed Q-learning algorithm

As remarked before, our system moves through various states based on the arrival and departure of flows as well as the actions we take at each decision timestamp. Let $Q\pi(s,a)$ be the Q-value associated with a state s and an action $\pi(s)$ specified by the policy π. The aim is to determine an optimal policy π^* such that it maximized the Q-value associated with a state.

$$\pi^*(s) = \arg\max_{a \epsilon A(S)} Q_\pi(s,a), \quad \forall s, \pi \qquad \textit{Equation 4. 12}$$

Let $a(n)$ be an update sequence that possesses the following properties,

$$\sum_{n=1}^{\infty} a(n) = \infty \; ; \; \sum_{n=1}^{\infty} (a(n))^2 = \infty \qquad \textit{Equation 4. 13}$$

Let b(n) be another update sequence that possesses the same properties as described in equation 4.13 along with the additional properties described below,

$$\sum_{n=1}^{\infty} \big(a(n)\big) + b(n)^2 < \infty; \quad \lim_{n \to \infty} \frac{b(n)}{a(n)} \to 0 \qquad \textit{Equation 4. 14}$$

At every decision epoch, we update the Q-value associated with only one state action pair and keep the rest of the values unchanged. This scheme can be represented by the following equation. Note that here a(n) is the learning rate parameter typically represented by α and γ is the discount factor.

$$Q_{n+1}(s,a) = (1 - ((a))Q_n(s.a) + a(n)[r(s,a) - l_n c(s,a) + \gamma max\, a' Q_n(s,a)] \quad \textit{Equation 4. 15}$$

Here *ln* is a Lagrange multiplier, which should be iterated along the timescale of b(n). Here *Sn* is given as a running sum of α raised to the power of iteration number, i.e., $Sn = 1.\alpha^0 + 1.\alpha + 0.\alpha^2 + \ldots$. The 1s and 0s are based on the number of switch exchanges on iteration *i*.

$$l_n = A[l_n + b(n)(S_n - S_{max})] \quad \textit{Equation 4. 16}$$

Here A is a projection operator to keep the Lagrange multiplier between [0, L] for a large L.

4.4.3.2 Q-learning implementation

We use a ε-greedy strategy to train the Q-network by investigating a random action with probability ε and exploiting an action with probability $1 - \varepsilon$ that maximizes the predicted long-term payoff. We acquire a new state and the resulting reward for each action we take. This is treated as a real-life experience that will be saved for later use. This procedure's algorithm is listed below.

Let's have a look at each of the steps we take in this algorithm.

- **Initialization:** We first initialize the Q-matrix to all zeros. The hyperparameters are set next. We set γ to 0.99, α to $\frac{1}{(n+1)^{0.6}}$, and $b(n)$ to $\frac{1}{n}$. We also initialize ε to 0.7. We then decrement this by 0.0025 per iteration till it reaches a minimum of 0.05. Finally, *Smax*, *Sn*, and *l*0 are initialized.
- **State updation:** For each iteration, we first update the state based on the incoming or outgoing flows. The state consists number of flows on each controller as mentioned earlier. Based on the state values the domain in migration and out migration domains will be identified using algorithms 1 and 2. Variations in the load rate disparity between two controllers within the migration matrix and the attributes of the switches being migrated within a single migration loop can lead to differences in the real-time reward value, denoted as R. In the subsequent phase, the migration matrix is examined to compute the return value for the migration triplet, and the in-

migration domain C_{in} with the highest calculated value is designated as the migration target for the out-migration domain C_{out} during this phase.

- **Decide action:** After the migration triplet is selected in the t-th iteration, the maximum valuation, denoted as $Q(C'_{out},C'_{in})_t$ of the next migration will be selected by the maximum greedy strategy.

  In the present state, the evaluation matrix Q is computed, and if it exceeds the original value, it indicates that the global migration optimal constraint has been met. If it does not surpass the initial value, another migration triplet will be selected instead.

  The migration triplet (Ci, Cj, Sk) is added to migration set A, and the candidate migration matrix's migration state is updated while it is removed from the migration set.

  Next, we make a choice between exploration and exploitation based on ε value in that current iteration. Based on this, we decide upon one of the 3 actions. Unless the action is *do nothing*, the next state is different from the current state. We make necessary updating and store the next state.

- **Reward calculation:** Now that we have the next state, we make calculations for reward, cost, and discounted reward. Update the Q-value for the current state-action pair. Finally, we set the current state as the next state and update ε value. Additionally in this step, we make a note of the cumulative discount reward, current discrete coefficient, and number of switch exchanges, which will be later necessary for making plots.

Algorithm 3: Load Balancing using Q-learning

input : *C*: set of all controllers

Ccluster: initial arrangement of switches and controllers

loadarray: load on switches for any iteration i

n(*s*): number of switches in the SDN environment

n(*C*): number of controllers in SDN environment

output: cumulative discounted rewards

discrete coefficients

number of switch exchanges between controllers

begin

Initialize the evaluation matrix Q with all 0, γ, α, $b(n)$, ε, *Smax*, *Sn*, and l_0.

foreach *timesteps* **do**

Determine the current system state (using last state and flows)

if *exploration phase* **then**

Choose one of the feasible actions at random

else

action =arg $\max_{a \epsilon A(S)} Q_\pi(s,a)$

Observe reward $r((s,a); l_n) = r(s,a) - l_n c(s,a)$

Go to next state s'

Update **Q(s,a)** according to equation 4.15

Update the LM according to equation 4.16

Set current state to next state (s ← s')

Record and update parameters for plotting graphs

if $\varepsilon > \varepsilon$-*min* **then**

ε = **ε** ∗**ε**-*dec*

else

ε = **ε**-min

CHAPTER FIVE

RESULTS AND DISCUSSIONS

5.1 Introduction

In this chapter, we will delve into a comprehensive exploration of the simulation environment employed to address the challenge of controller load imbalance. We will elaborate on the simulation tool utilized, the specific SDN controllers employed, the metrics employed to assess performance, and a thorough examination of the simulation results. Furthermore, we will critically evaluate these results and engage in a detailed discussion of our findings. To conclude, we will compare the performance of our proposed solution with two alternative strategies previously presented in related research.

5.2 Experiment Setup

A free jupyter notebook environment that runs on anaconda an open source python platform is used. To evaluate the performance of the proposed schemes, we use Python-based simulations. We use the Arnes network topology from the Internet Topology Zoo. Along with the topology, we make use of synthetic data generated from Python code, which fits with our assumptions regarding the influx and servicing of flows in switches.

For data generation, we come across the following 2 terms:

- **Arrival Rate (λ)** = we are assuming arrivals can be monitored using a Poisson process with parameter λ.
- **Service Rate (μ)** = we are assuming that the packet service times can be monitored using an Exponential process with parameter μ.

For each switch, we assume that the arrival is a Poisson process with parameter λi. However, the service rate parameter μ is taken for all switches. Since the arrivals occur as a Poi(λ) process, the inter-arrival times are exponentially distributed with mean λ. For each switch, we compute the time of arrival and add the service time to it to note the timestamp when the packets depart. It is interesting to observe that at each timestamp only one arrival or departure can occur. This means we can save tuples in the following format –

tuple(timestamp, switch, flow +/-)

Using these tuples we finally build our table of data. This is how our table will look roughly. For our experiment, we have taken 34 switches (from the Arnes topology) and 5 controllers.

Timestamp	Switch 01	Switch 02	Switch 03	...	Switch 34
0.2546	0	0	0	...	1
0.3947	1	0	0	...	0
0.4247	2	0	0	...	0
0.4547	3	0	0	...	0
...					
0.9649	0	0	1	...	0
0.9947	0	0	0	...	0

Figure 5. 1 Example of the Sample Data

5.2.1 Different Load scenario Experimental setup

We consider a system with $k = 5$ controllers and generate data for a variety of scenarios. To simulate real-world environments we consider the following cases:

- **Heavy Load:** This is the case where the load on switches is greater. This indicates that the switch arrival rate will be substantially higher than the service rate. To generate the data in this scenario, we examine the following rate of values.
  - λ = random numbers in the range (50,100)
  - μ = a random number in the range (1,5)
- **Light Load:** This is the case when the load on the switches is low. As a result, the arrival rate on the switches will be substantially lower than the service rate. To generate the data in this scenario, we examine the following rate of values.
  - λ = random numbers in the range (1, 5)
  - μ = a random number in the range (50, 100)
- **Skewed Load:** This is the case when the load on the switches is uneven. This indicates that the arrival rate on some switches will be significantly higher than on the other switches. Let's use a very high arrival rate on two randomly selected

switches for our purposes. To generate the data in this scenario, we examine the following rate of values.

- λ = high: random numbers in the range (50, 100); low: random numbers in range (5, 10)
- μ = a random number in the range (25, 50)

5.3 Analysis of the Results

In all scenarios, the total discounted reward eventually approaches a consistent value. This convergence usually occurs around the 400th iteration, indicating that Q-Learning is working properly. The cumulative discounted reward, which accounts for the sum of rewards over time while taking the discounting factor into account, finally reaches a stable and consistent value across different scenarios. After around 400 iterations of the learning process, the reward value converges. This milestone is crucial since it validates the effectiveness of the Q-Learning algorithm.

This convergence implies that the Q-Learning algorithm has learned optimal action strategies and has effectively adapted to the dynamics of the environment. It implies that the agent's decision-making process has settled into a consistent pattern, indicating a complete comprehension of how to traverse the environment to maximize its rewards while discounting the future consequences of its actions.

The Q-Learning algorithm exhibits its capacity to balance exploration and exploitation by refining its estimations of action values and progressively making better decisions over time by attaining this steady state. This accomplishment reinforces the algorithm's dependability and ability to guide decision-making in a variety of circumstances, strengthening its practical applicability in reinforcement learning tasks.

5.3.1 Result Analysis in Terms of Discrete Coefficient of Load Heavy

Figure 5.2 shows In the setting of a heavy load scenario, an interesting trend in the behavior of the discrete coefficient emerges. The discrete coefficient begins a discernible journey of steady decline as the system grapples with the weight of a considerable load. This decrease in the discrete coefficient value occurs throughout iterations or time steps, indicating a gradual decrease in its magnitude.

This decreasing trend in the discrete coefficient has significant implications. It denotes a significant juncture in the system's behavior—a point at which the system's load dynamics

change. As the discrete coefficient gradually decreases, it eventually stabilizes, remaining within a relatively limited and narrow range of values. This stabilization is a significant indicator that the system load has reached a condition of equilibrium and balance.

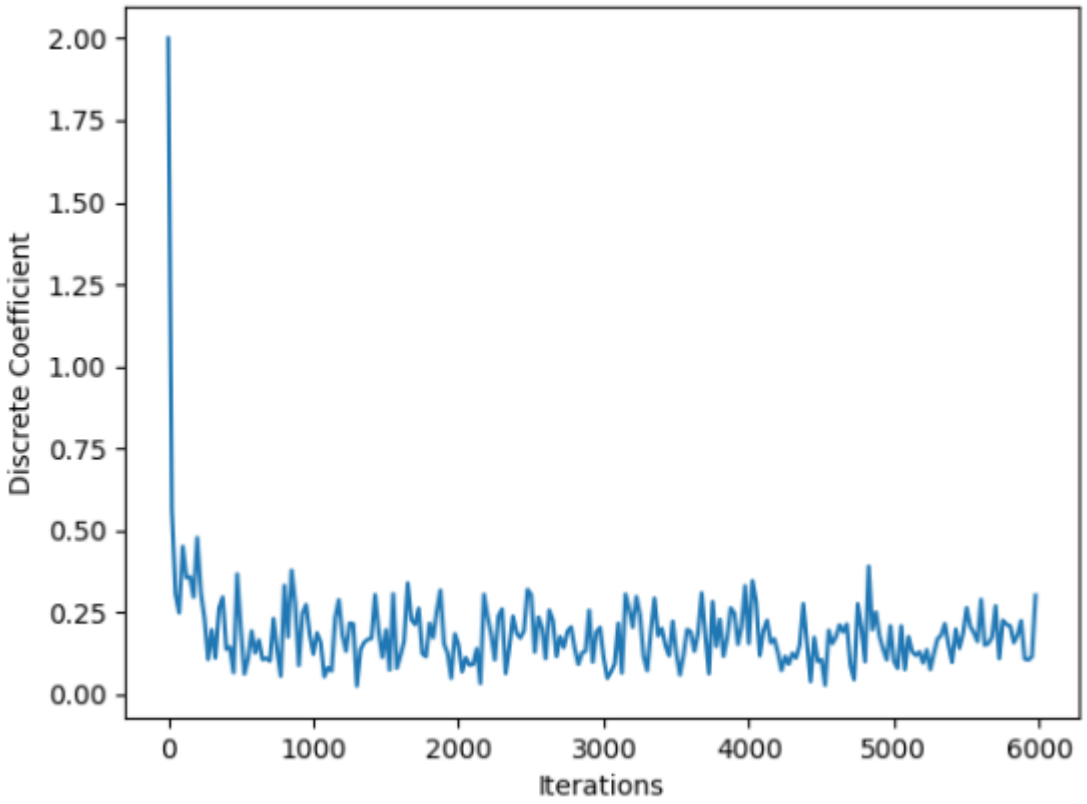

Figure 5. 2 Discrete coefficients of the system with time - Load Heavy

5.3.2 Result Analysis In Terms Of Cumulative Discounted Reward of Load Heavy

A fascinating pattern emerges in the context of a heavy load scenario as illustrated in Figure 5.3, as the cumulative discounted reward progressively converges to a value of around -55, a convergence that normally occurs over the course of around 400 iterations. Following this moment of convergence, all additional variations in the reward become so minor that they do not register as visually perceptible fluctuations on the curve's graphical depiction.

In this perspective, the convergence of the cumulative discounted reward at -55 is especially significant. It depicts an equilibrium point, a state of equilibrium that the system aims for despite the continual perturbations. This observed convergence demonstrates the system's adaptive capability. Despite the constant perturbations created by the dynamic environment, the system's decision-making mechanisms, driven by the cumulative discounted reward, display an exceptional capacity to navigate and readjust.

This stability indicates the system's decision-making framework's resilience and robustness. The system's capacity to maintain reasonably consistent performance despite constant change demonstrates its adaptability and the quality of its resource allocation algorithms.

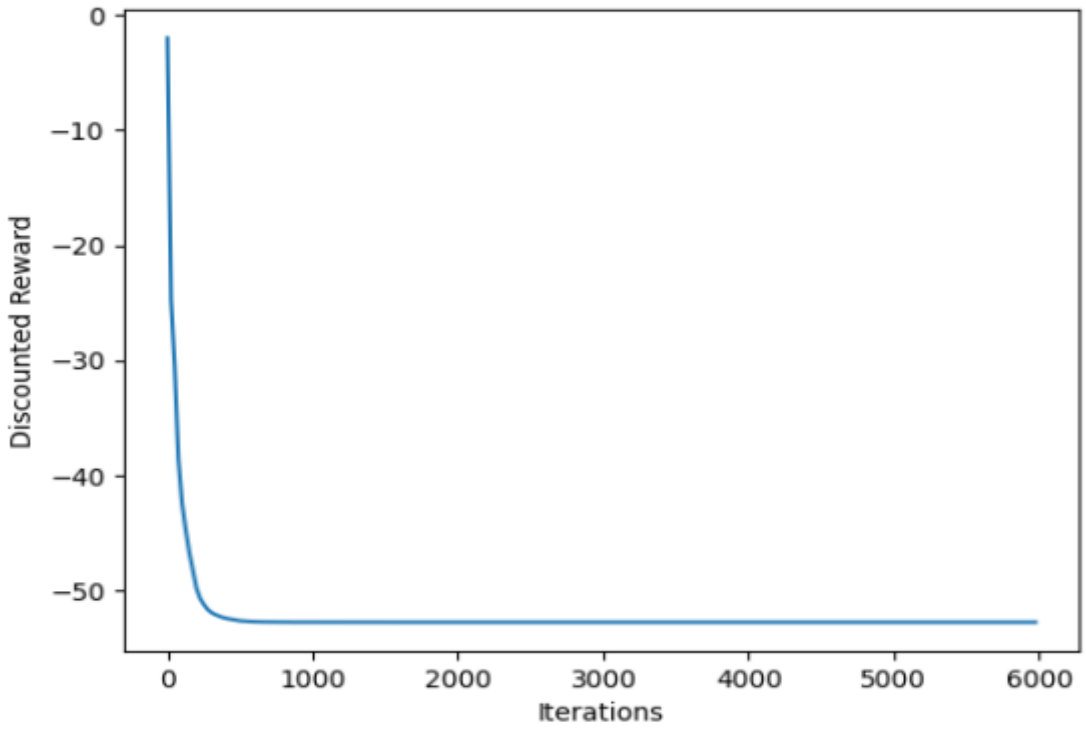

Figure 5. 3 Cumulative discounted reward of the system with time - Load Heavy

5.3.3 Result Analysis in Terms of Switch Exchanged for Load Heavy

In the case of a heavy load, as shown in Figure 5.4, the numerical count of switches exchanged follows a recognizable pattern, eventually converging around 4000. This figure is noticeably close to the predetermined upper limit, Smax, which was purposefully chosen.

The observed convergence of the numerical count of switches to a value near 4000 holds particular significance. It serves as a validation of the chosen value for Smax, indicating that this upper threshold was thoughtfully selected to accommodate the operational demands of the system under heavy load conditions. The fact that the actual switch count aligns closely with Smax suggests that the chosen limit effectively caters to the system's requirements, preventing excessive switch exchanges that might otherwise lead to performance degradation or instability.

Furthermore, this convergence serves as a practical benchmark, showcasing the system's adaptability and its capacity to gauge its operational needs. The convergence around 4000

implies that the system has effectively managed its state changes, utilizing the available resources while avoiding unnecessary switch exchanges beyond the chosen threshold.

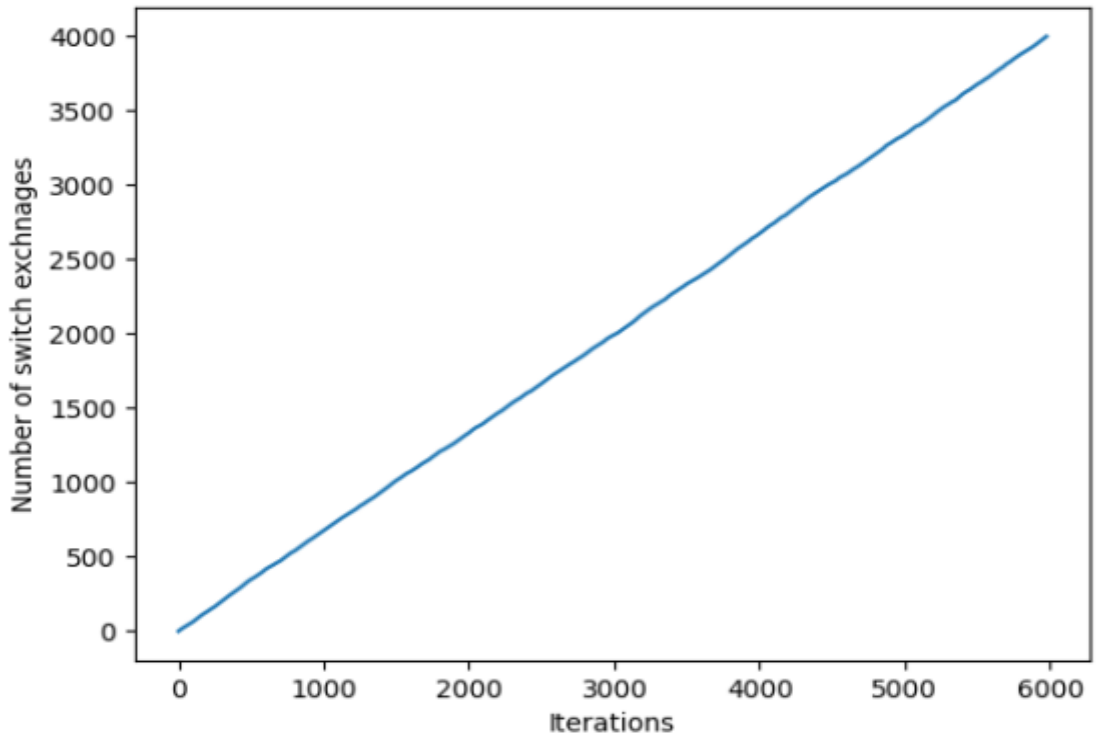

Figure 5. 4 Number of switches exchanged with time - Load Heavy

5.3.4 Result Analysis In Terms Of Discrete Coefficient for Load Light

Figure 5.5 shows the behavior of the discrete coefficient is noticeably different in the context of a light load scenario. The discrete coefficient, in particular, has a higher level of randomness and fluctuation. The larger fluctuations and less predictable patterns observed within the discrete coefficient values reflect this enhanced randomness. In contrast to situations with heavier loads or more balanced system conditions, where the discrete coefficient tends to exhibit more predictable trends and lower levels of variability, the presence of a light load appears to introduce uncertainty and volatility into the discrete coefficient's behavior. This phenomenon could be attributed to the system experiencing less stress or strain under a lighter load, resulting in a less constrained and more erratic behavior of the coefficient values.

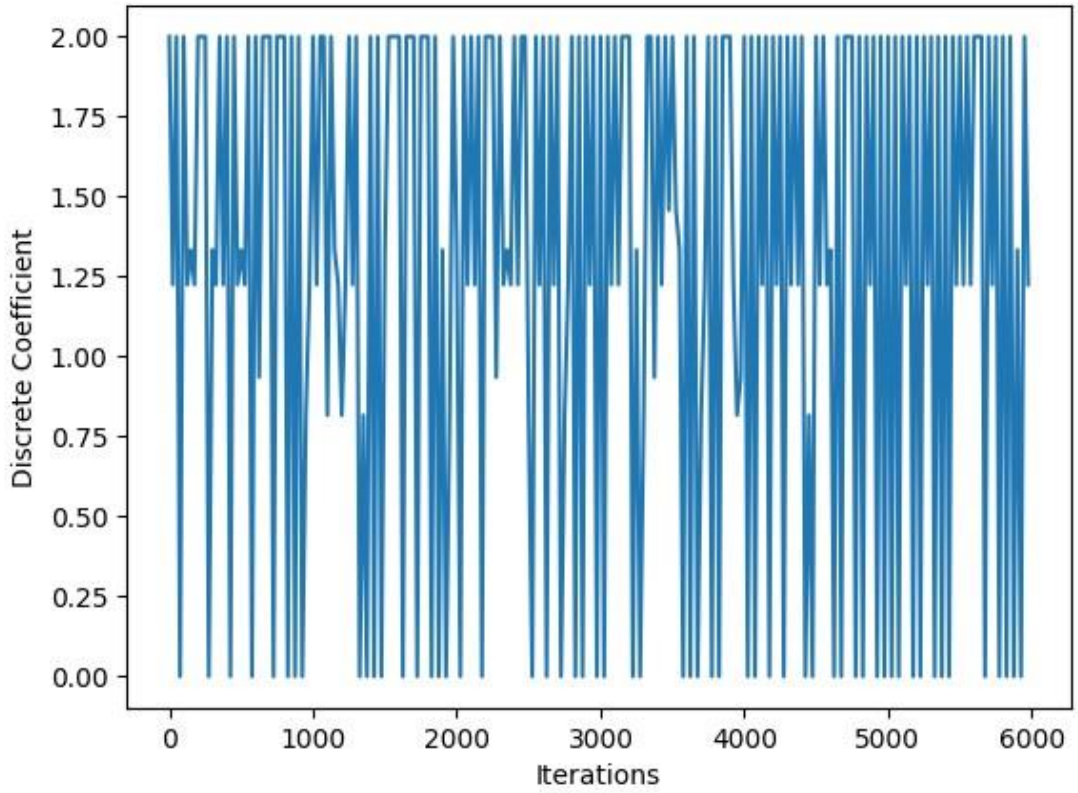

Figure 5. 5 Discrete coefficients of the system with time - Load Light

5.3.5 Result Analysis In Terms Of Discounted Reward for Load Light

Figure 5.6 shows in settings characterized by light load conditions, a distinct trend emerges when the cumulative discounted reward gradually approaches -115, which resulted in much more reduced discounted reward than heavy load and skewed load scenarios. This convergence normally occurs over the course of 400 repetitions. Following this point of convergence, any additional changes in the reward become so subtle that they fail to emerge as visually apparent variations when displayed on the curve's graphical depiction.

The minimal alterations seen beyond the moment of convergence highlight the system's resilience in the face of these ongoing shocks. This stability emphasizes the system's resilience and robustness, which are built into its decision-making architecture. The system's capacity to maintain consistent performance in the face of frequent change demonstrates its adaptability and the effectiveness of its resource allocation procedures.

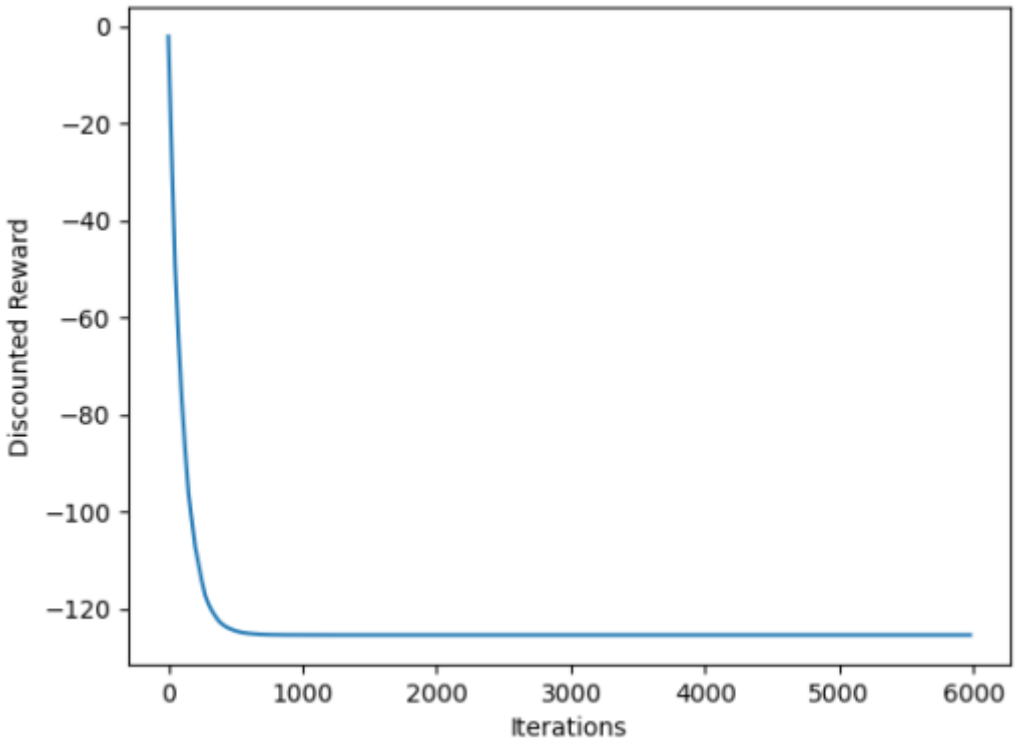

Figure 5. 6 Cumulative discounted reward of the system with time - Load Light

5.3.6 Result Analysis in Terms of Switch Exchanged for Load Light

Figure 5.7 shows that the cumulative representation of switches exchanged is plotted, allowing us to identify a distinct trend in connection to various load circumstances. The significantly smaller number of switches transferred in circumstances characterized by a light load is especially noticeable.

The cumulative plot of exchanged switches reveals a clear pattern: fewer switches are exchanged under light load conditions. This indicates that the system operates with greater stability and efficiency when the workload is lighter. The reduced need for frequent state changes suggests that the system adapts well to the workload, maintaining a more consistent configuration. This observation highlights the system's ability to optimize resource utilization and ensure smoother performance during periods of lower demand.

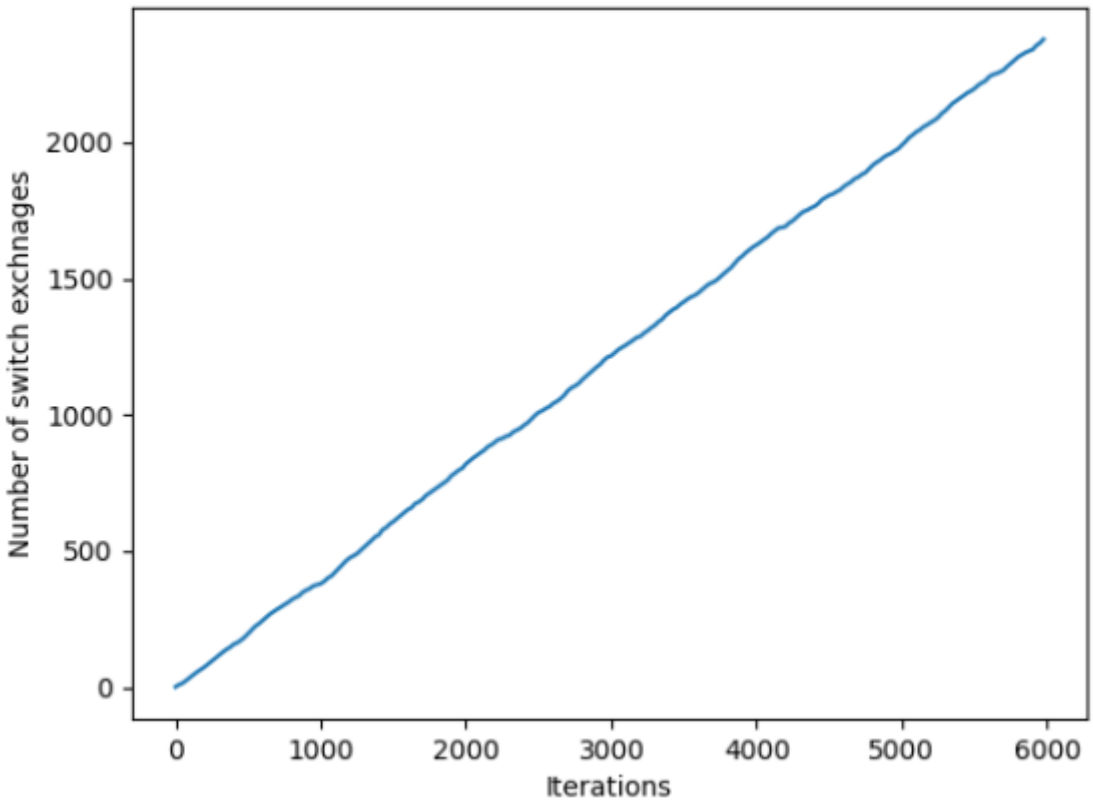

Figure 5. 7 Number of switches exchanged with time - Load Light

5.3.7 Result Analysis In Terms Of Discrete Coefficient for Load Skewed

Similarly, in a scenario with a skewed load distribution as displayed in Figure 5.8, a pattern similar to that seen in the heavy load condition occurs in relation to the discrete coefficient's behavior. The discrete coefficient, like heavy loads, follows a trajectory of slow decline, a trend that unfolds gradually through time or iterations.

The declining discrete coefficient values in this setting of skewed load distribution reflect a process of load adjustment inside the system. As the coefficient falls progressively, it finally reaches a point of equilibrium where it stabilizes, remaining present within a limited and relatively narrow range of values. This stabilization is critical because it indicates that the system's load dynamics have attained a condition of balance and constancy. The gradual decline and eventual stabilization of the discrete coefficient within a skewed load scenario parallel the characteristics observed in a heavy load context. This phenomenon underscores the system's ability to adapt and manage non-uniform workloads efficiently, demonstrating its capacity to achieve load equilibrium and effective resource allocation.

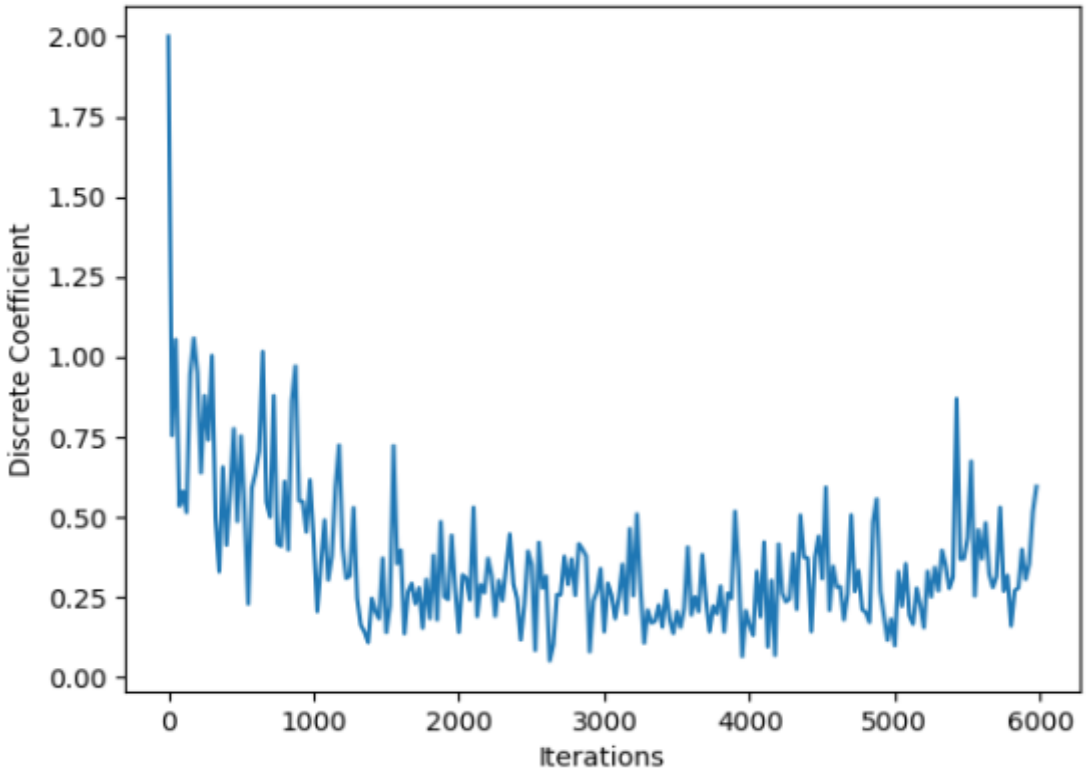

Figure 5. 8 Discrete coefficients of the system with time - Load Skewed

5.3.8 Result Analysis In Terms Of Discounted Reward for Load Skewed

Figure 5.9 shows that a different pattern appears inside scenarios with skewed load distributions when the cumulative discounted reward gradually converges towards a value of around -95. This convergence occurs over around 400 iterations. Following this point of convergence, subsequent variations in the reward become so tiny that they do not show up as detectable fluctuations on the curve.

The convergence of the cumulative discounted reward at around -95 is very significant in this context. It serves as an equilibrium point, representing a state of balance that the system strives for despite the constant perturbations. This convergence demonstrates the system's decision-making processes' flexibility. Despite the constant interruptions brought about by the dynamic environment, the system's capacity to navigate and adapt its methods based on the cumulative discounted reward is notable.

In conclusion, the convergence of the cumulative discounted reward around -95, followed by the extremely tiny variations, represents the system's ability to preserve equilibrium in the face of dynamic conditions. This stability and adaptability underscore the effectiveness

of the system's decision-making mechanisms, demonstrating consistent performance despite the ongoing obstacles posed by continuous arrivals, departures, and flow dynamics.

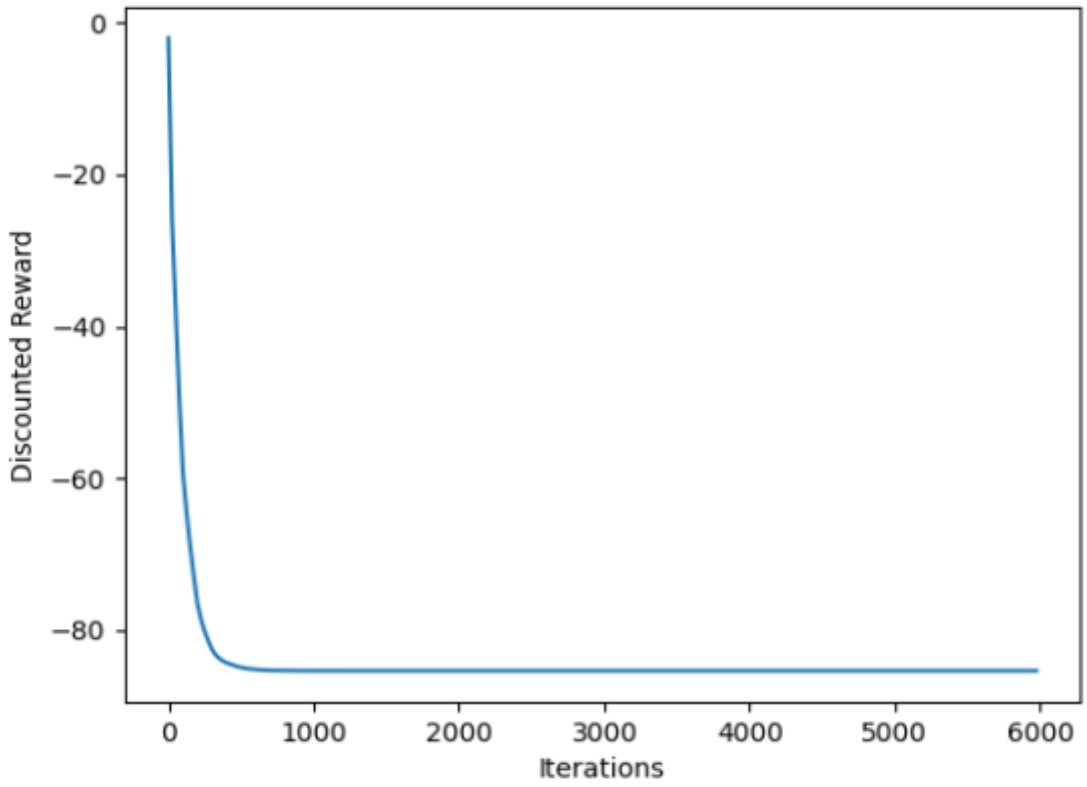

Figure 5. 9 Cumulative discounted reward of the system with time - Load Skewed

5.3.9 Result Analysis in Terms of Switch Exchanged for Load Skewed

Figure 5.10 displays that the cumulative plot of exchanged switches indicates a similar pattern to the observations recorded under heavy load. In this context, the cumulative numerical count of switches transferred is shown, revealing a trend that converges around 4000. This figure is very similar to the specified threshold, Smax, which has been carefully set to fit with the system's operational requirements, particularly in scenarios characterized by a heavy load.

The convergence of the numerical count of switches to around 4000 in this context is significant. It represents the system's harmonious equilibrium point, where the cumulative switch exchanges have stabilized around this value. This stabilization method emphasizes the system's responsiveness to shifting demands and its capacity to successfully manage its operating states.

The resemblance in behavior between the current and heavy load scenarios highlights a consistent operational principle. Converging switch exchanges around 4000, similar to

Smax, showcases the system's ability to optimize state transitions. This alignment of parameters and observed behavior underscores a thoughtful design, enhancing system adaptability and robustness across load variations.

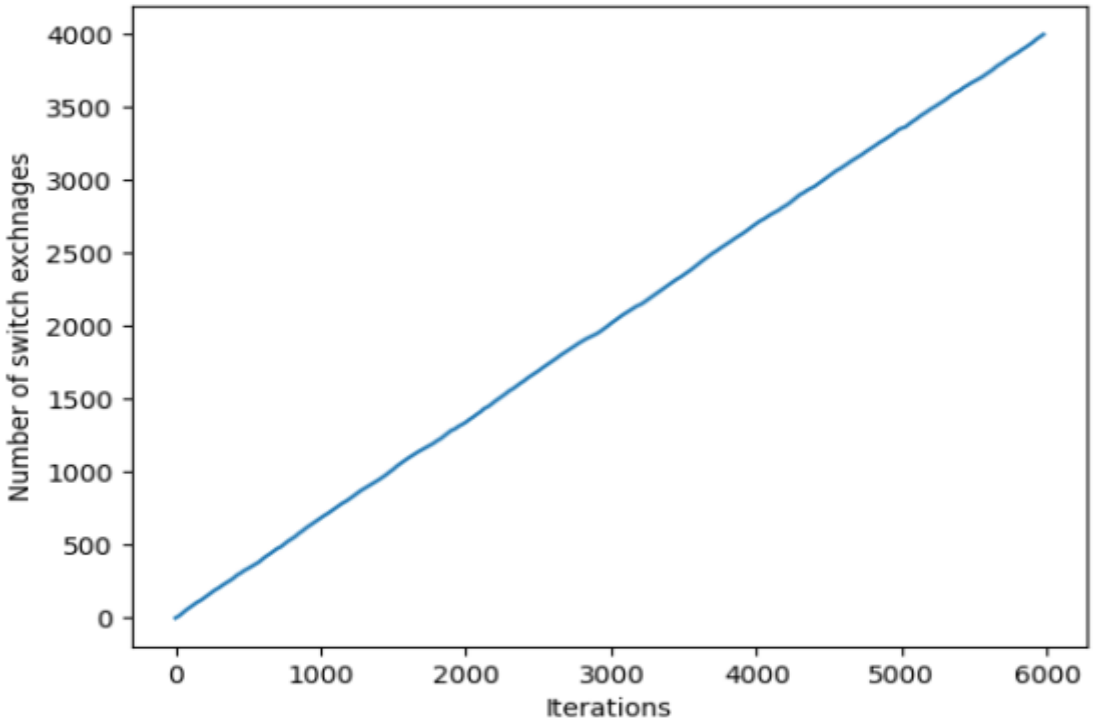

Figure 5. 10 Number of switches exchanged with time - Load Skewed

5.3.10 Comparison of the Work Load Distribution between Controllers over time

Figure 5.11 displays that Switch movement invariably introduces migration overhead into the system, the size of which is proportional to the number of migrations and controller-to-switch hops, and continuous switch migration increases migration overhead with the operation of the load balancing mechanism. Under the same network conditions of use, switch action conducted through various load balancing techniques will have a direct impact on the overhead of the switch migration, which includes the controller's traffic overhead and controller synchronization delay. Figure 5.11 below shows that the proposed mechanism has obvious advantages over switch migration load distribution among available controllers in multiple controller schemes.

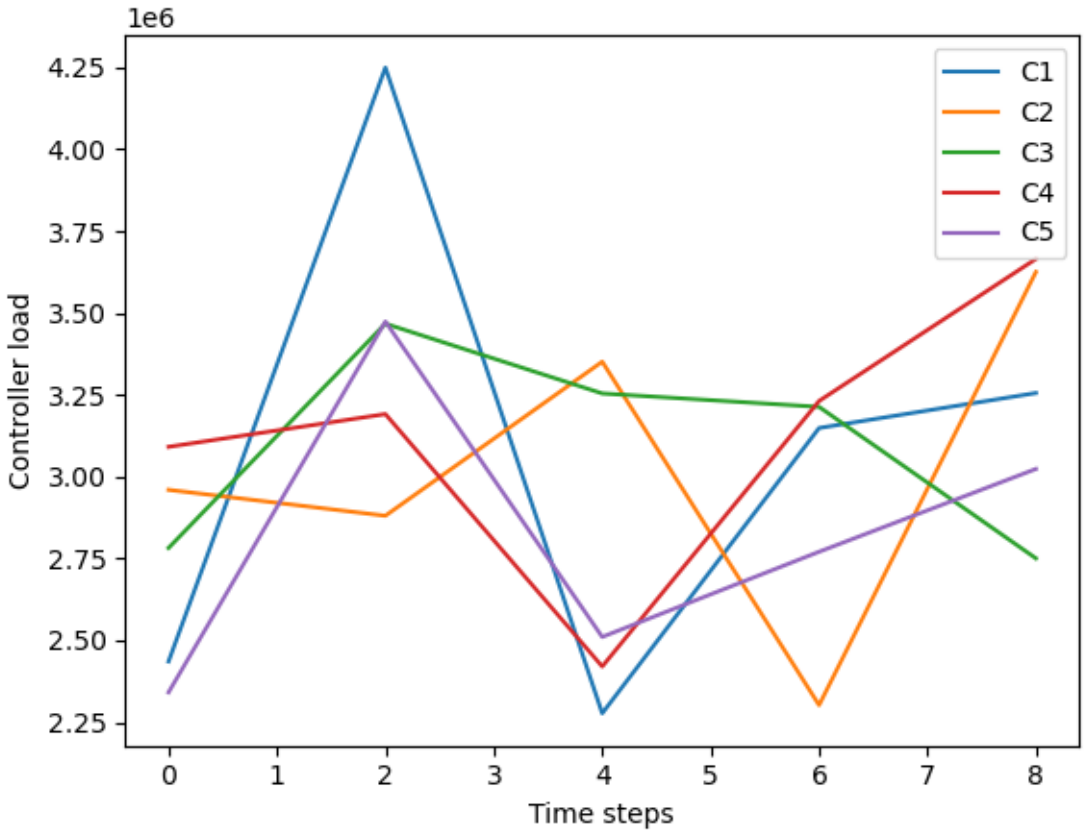

Figure 5. 11 Load distribution between controllers

5.3.11 Comparison of the performance Evaluation

A comparative experiment was added to the simulation experiment to completely analyze the performance of the switch migration mechanism described in this research as displayed in Figure 5.12. The experimental results show that the load balancing mechanism designed in this paper can effectively migrate the switch under the controller with trend overload to the relatively idle controller, which improves the utilization ratio of controllers' resources, reduces the response time of the switch's packet-in event, and improves the load balancing effect in the entire network environment.

When different load balancing systems are compared, the ratio in load balancing is a vital indicator to measure them, and each of them seeks to minimize the control plane's rate in load balancing. As it has been illustrated in the figure below the pace of load balancing of each controller is improved after employing the switch migration mechanism. As the packet-in message transmission rate increases, the value of the message is lowered due to the controller's capabilities restrictions.

Table 5.1 show the cumulative Discrete Coefficient of workload load distribution between proposed method and competing schemes in the simulation environments in skewed network condition.

Table 5. 1 Cumulative Discrete Coefficient result of different schemes

Methods	Cumulative Discrete Coefficient
SAR-LB	52.6456
RLB	58.31595
OQLB(proposed)	47.005

Since the lower the Discrete coefficient is the better the system achieves load balancing, The cumulative discrete coefficient of the proposed system shows better performance compared to other competing techniques were SAR-LB resulted in 52.6456, the proposed method improves it by 12% and improves the RLB by 19% were its cumulative discrete coefficient is 58.31595.

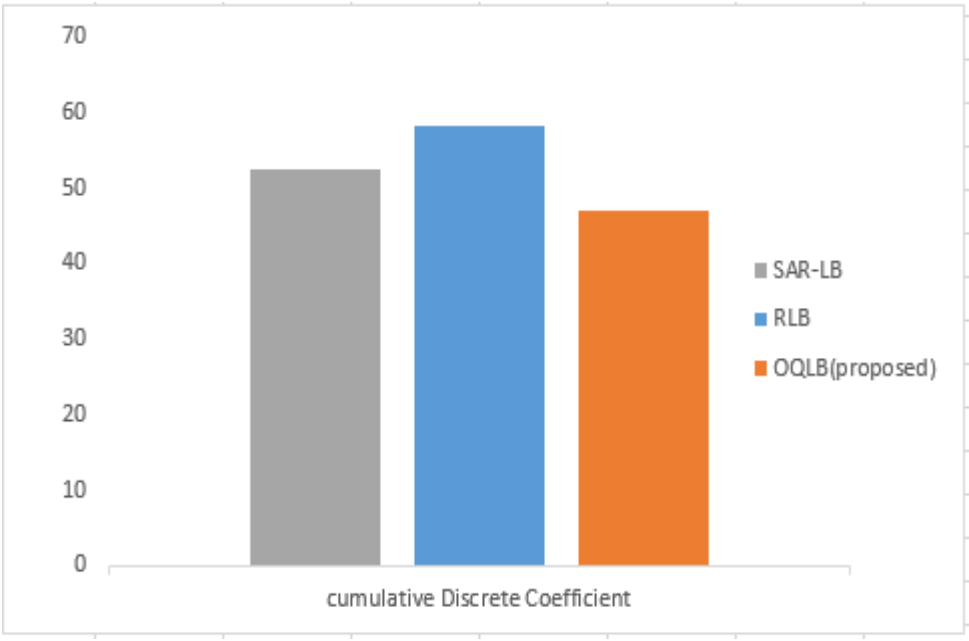

Figure 5. 12 Performance evaluation in terms of cumulative Discrete Coefficient

It can be seen from Figure 5.12 that the load balancing rate of the proposed mechanism is increased by 15.55% on average compared with the RLB and SAR-LB.

In Figure 5.13 as far as the total cumulative discrete coefficient of the system performs better than other competing schemes, the load balancing ratio of the system also achieves better compared to the other schemes.

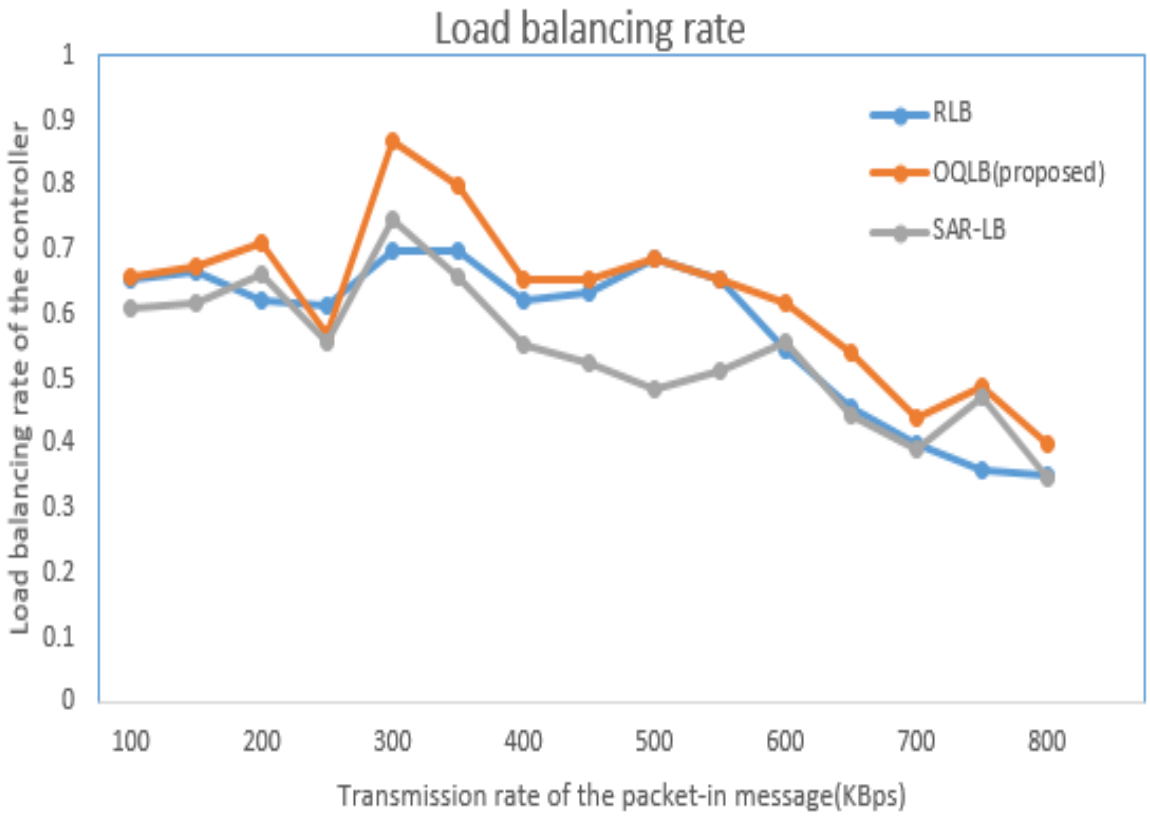

Figure 5. 13 Performance evaluation in terms of load balancing ratio

The load balancing rate of the proposed scheme was evaluated with competing schemes and shows a better load balancing rate compared to the others, were the load balancing rate defined in this study as follows:

$$\bar{R} = \sum_{i=1}^{n} R_{C_i}/n$$

Where Rci is the load ratio of each controller and n is the total number of controllers used in this study. Hence, the proposed method shows a better load balancing rate than the other competing schemes.

5.4 Discussions

The Q-Learning algorithm exhibits its capacity to balance exploration and exploitation by refining its estimations of action values and progressively making better decisions over time by attaining this steady state. This accomplishment reinforces the algorithm's dependability and ability to guide decision-making in a variety of circumstances, strengthening its practical applicability in reinforcement learning tasks.

- The discrete coefficient is much more random and higher in the case of light load than in the case of heavy and skewed load as shown in Fig 5.3. In the latter cases, we notice that the discrete coefficient gradually decreases until it stays roughly in a small range, which means that the system load has stabilized and is equally distributed.
- For the heavy load scenario, we see in Figure 5.2 that the discrete coefficient is in the range of 0.25-0.5 roughly. This is a good value since the system is constantly being disturbed by new arrivals and departures or flows. We also see that the cumulative discounted reward converges to -55 roughly around 400 iterations. After that, the change is very small to be visible on the curve.
- For the light load scenario, we see that the discrete coefficient is random. This is justified by the fact that the discrete coefficient here is a measure of the deviation of the number of flows in each switch. Since flows are serviced very quickly, the system usually stays in a state where the load is on a few controllers only and the others have 0 load. This gives a large D value.
- For the skewed load scenario in Fig 5.3, we observe that the discrete coefficient values are more random than the heavy load scenario. This is also backed by the fact that in the case of skewed load, inflow is constantly more on a few switches and much less on others.
- Figure 5.11 shows the workload distribution between controllers, and it signifies that through the implementation of the proposed Q-learning algorithm, the workload is being distributed among available controllers through time steps.

5.5 Summary

RQ1: How to assess controller load imbalance and decide whether to conduct switch migration using a Reinforcement learning technique?

The first question of the research raises, How to assess controller load imbalance and decide whether to conduct switch migration using a Reinforcement learning technique. To answer this question we have tested our system in different load scenarios such as heavy load scenarios, light load scenarios, and skewed load scenarios. The discrete coefficient between controllers has been used as a measure of the degree of load imbalance. And also switch migration efficiency was used to select the switch to be migrated. As a result, the implemented technique shows that the discrete coefficient stays roughly in a small range and the cumulative discounted reward of the system also shows there is an undiscernible cumulative reward between controllers which signifies, there is a balanced load distribution between controllers.

RQ2: How can we enhance current approaches to load balancing among controllers using a reinforcement learning algorithm?

The second question raised in this research has been answered when We make use of an online ε-greedy based Q-learning which makes the reward cost optimization along with the exploration and exploitation strategy. Based on the result, we can claim that our method improves load balancing in the dynamic environment of SDN.

RQ3: How to compare the performance of the newly proposed scheme against existing methods?

The load balancing rate of different existing works has been compared with our online Q-learning framework. We have compared our work with RLB and SAR-LB and found that our newly proposed solution performs better in terms of load balancing rate compared to the other two mechanisms. The result shows that the newly proposed approach improves the load balancing rate by 15.55% on average.

CHAPTER SIX

CONCLUSION AND FUTURE WORK

6.1 Conclusions

The problem of controller load imbalance between multiple controllers in SDN is caused by imbalanced load distribution among the controllers and static mapping between switch and controller. As a result, certain controllers have become overloaded while others are underloaded. We presented our design of a switch migration method with RL-based switch migration decision-making in this research, which delivers more evenly balanced load distribution between dispersed SDN controllers than existing schemes.

The conducted study has demonstrated an approach to load balancing, focusing on the utilization of switch migration as a means to achieve a more efficient distribution of workloads. Unlike the conventional method that relies on threshold values, which has been the predominant approach in various prior research endeavors, this study introduces a significant departure by employing discrete coefficients as the measurement parameter.

Traditionally, load balancing has been a critical concern in distributed computing environments to ensure that computational resources are optimally utilized and that no single node or component becomes overly burdened. Many previous research works have centered their efforts around setting specific threshold values, which act as benchmarks to trigger load migration from heavily loaded nodes to less utilized ones. While this threshold-based strategy has provided reasonable outcomes, it has its limitations and might not adapt optimally to dynamic changes in system conditions.

In this context, the current study seeks to enhance load balancing effectiveness by introducing discrete coefficients as a more adaptable measurement parameter. These coefficients offer a more nuanced and responsive approach to load assessment. Rather than relying on fixed thresholds, which may not effectively capture the intricacies of the system's real-time performance, discrete coefficients provide a graduated scale of measurement. This enables load balancers to make more informed decisions about when and how to migrate workloads, promoting a finer degree of control over the distribution process.

By embracing this Q-learning approach, the study aims to address some of the limitations of traditional threshold-based load balancing techniques. The discrete coefficients allow for a

more finely tuned response to workload fluctuations and variations in system conditions. Consequently, the load balancing mechanism can provide a smoother and more balanced distribution of computational tasks, ultimately contributing to improved system performance and resource utilization.

In summary, the study's contribution lies in its innovative use of discrete coefficients as a measurement parameter for load balancing through switch migration. By departing from the established practice of relying solely on threshold values, this approach offers a more adaptive and refined way to manage workloads in distributed computing environments. The potential benefits include better responsiveness to dynamic changes, enhanced load distribution accuracy, and overall improved system efficiency.

6.2 Major Contribution

The following are the primary contributions of this thesis work:

- This paper investigates controller load balancing based on switch migration.
- This mainly focused on altering the weakness of threshold based load balancing techniques and employed the discrete coefficient between controllers for the judgment of controller load overloading with the use of an online Q-learning algorithm.
- This research expands on previous switch migration work by proposing a Q-learning approach to balance controller loads across multiple controllers.
- The proposed method outperforms better than the selected schemes in terms of cumulative discrete coefficient where it shows a 19 % improvement to RLB and a 12 % percent improvement to SAR-LB.
- This work achieved a better load balancing rate compared with the RLB and SAR-LB methods and shows an improvement of 15.55% load balancing rate.

6.3 Future Work

The findings have inspired us to continue working on this research. Our next goal would be to work on other metrics such as throughput, response time, and packet delay, as well as to investigate other alternatives such as deep Q-learning to capture network traffic behavior to balance the controller and efficient switch migration in dynamic networks.

SPECIAL ACKNOWLEDGMENT

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

ASTU/SM-R/832/23

Adama, Ethiopia

References

Amin, R., Reisslein, M., & Shah, N. (2018). Hybrid SDN networks: A survey of existing approaches. *IEEE Communications Surveys and Tutorials*, *20*(4), 3259–3306. https://doi.org/10.1109/COMST.2018.2837161

Amin, R., Rojas, E., Aqdus, A., Ramzan, S., Casillas-Perez, D., & Arco, J. M. (2021). A Survey on Machine Learning Techniques for Routing Optimization in SDN. *IEEE Access*, *9*, 104582–104611. https://doi.org/10.1109/ACCESS.2021.3099092

Begam, G. S., Sangeetha, M., & Shanker, N. R. (2022a). Load Balancing in DCN Servers through SDN Machine Learning Algorithm. *Arabian Journal for Science and Engineering*, *47*(2). https://doi.org/10.1007/s13369-021-05911-1

Begam, G. S., Sangeetha, M., & Shanker, N. R. (2022b). Load Balancing in DCN Servers through SDN Machine Learning Algorithm. *Arabian Journal for Science and Engineering*, *47*(2), 1423–1434. https://doi.org/10.1007/s13369-021-05911-1

Beiruti, M. A., & Ganjali, Y. (n.d.). Migration Scheduling in Distributed SDN Controllers. *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, 1–2.

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

Cello, M., Xu, Y., Walid, A., Wilfong, G., Chao, H. J., & Marchese, M. (2017). BalCon: A distributed elastic SDN control via efficient switch migration. *Proceedings - 2017 IEEE International Conference on Cloud Engineering, IC2E 2017*, 40–50. https://doi.org/10.1109/IC2E.2017.33

Chen, H., Cheng, G., & Wang, Z. (2016). A game-theoretic approach to elastic control in software-defined networking. *China Communications*, *13*(5), 103–109. https://doi.org/10.1109/CC.2016.7489978

Cheng, G., Chen, H., Wang, Z., & Chen, S. (2015). DHA: Distributed decisions on the switch migration toward a scalable SDN control plane. *Proceedings of 2015 14th IFIP Networking Conference, IFIP Networking 2015*. https://doi.org/10.1109/IFIPNetworking.2015.7145319

Dixi, A., Hao, F., Mukherjee, S., Lakshman, T. V., & Kompella, R. R. (2014). ElastiCon:

An elastic distributed SDN controller. *ANCS 2014 - 10th 2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 17–27. https://doi.org/10.1145/2658260.2658261

Dotan, D., & Pinter, R. Y. (2005). HyperFlow: An integrated visual query and dataflow language for end-user information analysis. *Proceedings - 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, 2005*, 27–34. https://doi.org/10.1109/VLHCC.2005.45

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

Etengu, R., Tan, S. C., Kwang, L. C., Abbou, F. M., & Chuah, T. C. (2020). AI-assisted framework for green-routing and load balancing in hybrid software-defined networking: Proposal, challenges and future perspective. In *IEEE Access* (Vol. 8, pp. 166384–166441). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2020.3022291

Ethem, A. (n.d.). *Ethem Alpaydin-Introduction to Machine Learning-The MIT Press (2014)*.

Ethio Telecom's data and internet users Ethiopia 2021 | Statista. (n.d.). Retrieved September 29, 2023, from https://www.statista.com/statistics/749569/ethiopia-ethio-telecom-data-and-internet-subscribers/

Faezi, S., & Shirmarz, A. (2023). A Comprehensive Survey on Machine Learning using in Software Defined Networks (SDN). *Human-Centric Intelligent Systems*, *3*(3), 312–343. https://doi.org/10.1007/s44230-023-00025-3

Feamster, N., Rexford, J., & Zegura, E. (2014). The road to SDN. *ACM SIGCOMM Computer Communication Review*, *44*(2), 87–98. https://doi.org/10.1145/2602204.2602219

Filali, A., Cherkaoui, S., & Kobbane, A. (2019). Prediction-Based Switch Migration Scheduling for SDN Load Balancing. *IEEE International Conference on Communications, 2019-May*, 1–6. https://doi.org/10.1109/ICC.2019.8761469

Foundation, O. N. (2012). OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04). *Current*, *0*, 1–36.

Fu, Y., Bi, J., Wu, J., Chen, Z., Wang, K., & Luo, M. (2014). A dormant multi-controller model for software defined networking. *China Communications*, *11*(3), 45–55. https://doi.org/10.1109/CC.2014.6825258

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. *7th Python in Science Conference (SciPy 2008), SciPy*, 11–15.

Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D. C., & Gayraud, T. (2014). Software-defined networking: Challenges and research opportunities for future internet. In *Computer Networks* (Vol. 75, Issue PartA, pp. 453–471). Elsevier B.V. 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. In *Journal of Network and Computer Applications* (Vol. 174). Academic Press. https://doi.org/10.1016/j.jnca.2020.102856

Han'guk T'ongsin Hakhoe., & Han'guk Chŏnja T'ongsin Yŏn'guso. (2012). *ICTC 2012 : 2012 International Conference on ICT Convergence : "Global Open Innovation Summit for Smart ICT Convergence," October 15-17, 2012, Ramada Plaza Jeju Hotel, Jeju Island, Korea*. IEEE.

Hassas Yeganeh, S., & Ganjali, Y. (2012). Kandoo: A framework for efficient and scalable offloading of control applications. *HotSDN'12 - Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks*, 19–24. https://doi.org/10.1145/2342441.2342446

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, F., Hao, Q., & Bao, K. (2014). A survey on software-defined network and OpenFlow: From concept to implementation. *IEEE Communications Surveys and Tutorials*, *16*(4), 2181–2206. https://doi.org/10.1109/COMST.2014.2326417

Hu, T., Guo, Z., Yi, P., Baker, T., & Lan, J. (2018). 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*. 452–464.

Hu, T., Zhang, J., Wang, L., & Qiao, D. (2017). *An Improved Switch Migration Approach to Controller Load Balancing in SDN*.

Hu, Y., Wang, W., Gong, X., Que, X., & Cheng, S. (2013). BalanceFlow: Controller load

balancing for OpenFlow networks. *Proceedings - 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, IEEE CCIS 2012*, *2*, 780–785. https://doi.org/10.1109/CCIS.2012.6664282

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

Jingmei Li;Xinliang Hu;Mengqi Zhang. (2018). *Proceedings of 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC 2018) : October 12-14, 2018, Chongqing, China*.

Jiru, M. A., Adere, K., Science, A., Krishna, T. G., Science, A., Perumalla, J. R., & Science, A. (2023). *An Improved Switch Migration Method- Based Efficient Load Balancing for Multiple Controllers in Software-Defined Networks*. *25*(1), 1–21. https://doi.org/10.4018/JCIT.326136

Kidist, M. (2022). *Improving the Performance of Software-Defined Network Load Balancer Using Open Flow Based Multi-Controller Topology*.

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, Z., Zhou, X., Gao, J., & Qin, Y. (2019). SDN Controller Load Balancing Based on Reinforcement Learning. *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, *2018-Novem*, 1120–1126. https://doi.org/10.1109/ICSESS.2018.8663757

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

Lin, Y. D., Lin, P. C., Yeh, C. H., Wang, Y. C., & Lai, Y. C. (2015). An extended SDN architecture for network function virtualization with a case study on intrusion prevention. *IEEE Network*, *29*(3), 48–53. https://doi.org/10.1109/MNET.2015.7113225

Quach, H.-N., Yoem, S., & Kim, K. (2020). *Survey on Reinforcement Learning based Efficient Routing in SDN CCS CONCEPTS • Networks → Network protocols → Network layer protocols → Routing protocols*.

Rojas, E., Doriguzzi-Corin, R., Tamurejo, S., Beato, A., Schwabe, A., Phemius, K., &

Guerrero, C. (2018). Are we ready to drive software-defined networks? A comprehensive survey on management tools and techniques. *ACM Computing Surveys*, *51*(2). https://doi.org/10.1145/3165290

Rupani, K., Punjabi, N., Shamdasani, M., & Chaudhari, S. (2020). *Dynamic Load Balancing in Software-Defined Networks Using Machine Learning*. 283–292. https://doi.org/10.1007/978-981-15-0790-8_28

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

Selvi, H., Gur, G., & Alagoz, F. (2016). Cooperative load balancing for hierarchical SDN controllers. *IEEE International Conference on High Performance Switching and Routing, HPSR*, *2016-July*, 100–105. https://doi.org/10.1109/HPSR.2016.7525646

Shirmarz, A., & Ghaffari, A. (2020). Performance issues and solutions in SDN-based data center: a survey. In *Journal of Supercomputing* (Vol. 76, Issue 10). Springer US. https://doi.org/10.1007/s11227-020-03180-7

Shtykh, R. Y., & Suzuki, T. (2014). Distributed data stream processing with onix. *Proceedings - 4th IEEE International Conference on Big Data and Cloud Computing, BDCloud 2014 with the 7th IEEE International Conference on Social Computing and Networking, SocialCom 2014 and the 4th International Conference on Sustainable Computing and C*, 267–268. https://doi.org/10.1109/BDCloud.2014.54

Sufiev, H., & Haddad, Y. (2017). A dynamic load balancing architecture for SDN. *2016 IEEE International Conference on the Science of Electrical Engineering, ICSEE 2016*. https://doi.org/10.1109/ICSEE.2016.7806104

Sun, P., Guo, Z., Wang, G., Lan, J., & Hu, Y. (2020). MARVEL: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning. *Computer Networks*, *177*. https://doi.org/10.1016/j.comnet.2020.107230

Verma, S., Sharma, R., Deb, S., & Maitra, D. (2021). Artificial intelligence in marketing: Systematic review and future research direction. *International Journal of Information Management Data Insights*, *1*(1). https://doi.org/10.1016/j.jjimei.2020.100002

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*, 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*. https://doi.org/10.1016/j.comnet.2018.05.012

Wang, Y., Yu, J., Pei, K., & Qiu, X. (2017a). 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

Wang, Y., Yu, J., Pei, K., & Qiu, X. (2017b). 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

Xiang, M., Chen, M., Wang, D., & Luo, Z. (2022). Deep Reinforcement Learning-based load balancing strategy for multiple controllers in SDN. *Electronics and Energy*, *2*, 100038. https://doi.org/10.1016/j.prime.2022.100038

Ye, X., Cheng, G., & Luo, X. (2017). Maximizing SDN control resource utilization via switch migration. *Computer Networks*, *126*, 69–80. https://doi.org/10.1016/j.comnet.2017.06.022

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). https://doi.org/10.3390/electronics10020162

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