Addis Ababa University

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

Telecommunication Engineering Graduate Program

Thesis on

Multi-level Interval Based Load Balancing for Multi-controller SDN

By

Amanuel Bekele Bayu

Advisor

Surafel Lemma Abebe (Ph D)

Submission date: October 02, 2023

Addis Ababa University

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

Telecommunication Engineering Graduate Program

This is to certify that the thesis prepared by Amanuel Bekele Bayu, entitled Multi-level Interval Based Load Balancing for Multi-controller SDN and submitted in partial fulfillment of the requirements for the degree of Master of Science Telecommunication Engineering complies with the regulations of the university and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Internal examiner:…… Signature:…… Date:……..

External examiner:…… Signature:…… Date:……..

Adviser: Surafel Lemma (PhD) Signature:…… Date:……..

Dean, School of Electrical and Computer Engineering

Dedication

This research work is dedicated to my families (my wife Ayantu and my son Bekam and also my mother, my sisters and my brother).

Abstract

Software Defined Networking (SDN) is a new approach of managing and programming networks that is managed by a centralized controller. For a larger network, single controller architecture will be inefficient to manage the network because the increase in the network traffic causes network congestion and transmission delays. To tackle this issue, software defined architecture with multiple controllers has been introduced in recent researches. However, this strategy introduces a new problem of unequal load balance on the distributed SDN controller. To address this issue, several load balancing approaches have been developed. But most of them are domain specific and have relatively high migration cost, communication overhead and packet loss with low load balancing rate. An Improved Switch Migration Decision Algorithm (ISMDA) is one of the algorithm that solves the SDN load imbalance when the incoming data load is high traffic (traffic load greater than 500 p/s). The under loaded controller is selected by the load balancing module among the controller group by estimating variance and the average load status of the controllers. It improves controller throughput, migration cost and response time by running different modules like load judgment, switch selection and controller selection module. But it splits the load only into two intervals, i.e., overloaded and under-loaded, which limits its capability to distinguish the controller load status accurately. This paper proposes Multi-level Interval Based Load Balancing (MIBLB) algorithm that modifies the ISMDA algorithm by implementing a multi-level load interval, i.e., idle, normal, overloaded and highly overloaded, and applies dynamic threshold for switch migration. Dividing the load status into multi-level intervals enables accurate migration by giving priority to highly overloaded controller, and avoid congestion and packet loss. The efficiency will be evaluated in terms of network performance evaluation parameters, i.e., throughput, response time, and average number of migrations. The experimental results show that MIBLB reduces number of migrations and controller response time of ISDMA by 6.43% and 3.53%, respectively. The controller throughput of ISDMA is also increased by 6.82% while sustaining efficient load balancing rate.

Keywords: *Load balancing algorithm, Software defined networking, multi-controller, switch migration.*

Acknowledgements

First, I would like to thank the almighty God for giving me the opportunity to accomplish this work. Second, I would like to express my sincere gratitude to my advisor, Dr. Surafel Lemma, for his support, guidance and encouragement throughout this work. Third, I express my very profound gratitude to my family and friends for their continuous support, prayers, and encouragement. Last but not least, I also want to extend my appreciation to my company Ethio telecom and Addis Ababa Institute of Technology (AAiT) for opening such an opportunity to study MSc program.

Table of Contents

List of Figures

List of tables

Acronyms

bps	Bit per second
CALB	Controller Adaption and Migration Decision
DALB	Dynamic and Adaptive Load Balancing
IoT	Internet of things
ISMDA	Improved Switch Migration Decision Algorithm
MIBLB	Multi-level interval based load balancing
Kbps	Kilo byte per second
LB	Load Balancing
LBR	Load Balancing Rate
Mc	Migration Cost
OF	Open Flow
ODL	Open Day Light
QoS	Quality of Service
SDN	Software Defined Networking
SL	Switch Load
SARSA	State Action Reward State Action
TCP	Transmission Control Protocol

1. Introduction

Software-Defined Networking is a novel network paradigm that enables flexible management of networks [1]. Unlike traditional networks, SDN separates the control plane from the data plane in network devices such as switches and routers [2]. This new concept suggests the use of a centralized controller that determines the behavior of all forwarding components in the network. Southbound interfaces permit communication between the control plane and the data plane, while northbound interfaces provide enormous possibilities for networking programmability, like creating applications that can automate all networking tasks.

As the network size increases, the single centralized controller cannot meet the increasing demand for flow processing [3] . A single centralized controller may not be able to handle many requests for flow processing which results in controller performance degradation and eventually a dysfunctional SDN-enabled network. A practical solution would be the deployment of a multi-controller architecture where a group of controllers collaboratively handle a massive amount of network traffic and flow processing requests [4]. However, this approach introduces a new problem of unequal load balance on the distributed SDN controller [5]. To avoid load imbalance issues, several types of load balancing approaches are designed for multi-controller SDN. But most of them are domain specific and have relatively high migration costs, communication overhead and packet loss with a low load balancing rate.

The major contribution of this paper is to modify one of the load balancing algorithms, Improved Switch Migration Decision Algorithm (ISMDA). ISMDA [6] overcomes the network problem when the incoming load is high traffic (elephant flow). Unlike other algorithms, ISMDA uses the controller variance and mean load status to identify the set of under-loaded controllers in the SDN environment. But ISMDA is not designed in such a way that it is capable of distinguishing the controller load status accurately. This will lead to low performance, uneven load balance and inaccurate load status.

In the ISMDA, even though the algorithm use load variance and migration efficiency to balance the load in the SDN network, it has limitations due to the use of two load categories, i.e., under-loaded and overloaded. Because of improper load interval division, the ISMDA approach neither maintains load balancing efficiency nor fully utilizes the controller's resources. This leads to inefficient load balance because controllers that reach bottlenecks and controllers that

exceed the threshold by even one packet will be treated equally. This implies that there is no chance to give priority to a controller that is highly overloaded. The scheme also did not apply the dynamic threshold in the case when all controllers in the network get overloaded. When this situation happens, a packet will be rejected. These drawbacks make the algorithm less efficient and cause high packet loss.

To mitigate the aforementioned drawbacks, Multi-level Interval Based Load Balancing (MIBLB) algorithm is proposed. The MIBLB modifies the ISMDA algorithm by implementing a multi-level load interval to shift the controller's load in advance to prevent packet rejection and to reduce the migration cost during the load balancing. Additionally, it implements the dynamic threshold to avoid bottlenecks and packet loss.

1.1. Statement of the problem

The network congestion and SDN controller overloading have become a serious problem in most of the SDN environments due to the increasing of network traffic [7]. Distributed SDN is a promising approach because of their scalable and reliable deployments in SDN environments. However, this approach introduces a new problem of unequal load balance on the distributed SDN controller [5]. To mitigate this issue, several types of load balancing approaches such as reinforcement learning LB mechanism (RL-LBM), improved ant colony algorithm, efficient switch migration based load balancing (ESMLB), controller adaption and migration decision (CAMD) and dynamic and adaptive load balancing (DALB) are proposed for multi-controller SDN. But most of them are domain specific, have high migration cost, communication overhead, and packet loss with low load balancing rate [8], [9].

An Improved Switch Migration Decision Algorithm (ISMDA) [6] is one of the algorithms that solves the network problem when the incoming load is high traffic. Unlike other algorithms, ISMDA uses the controller variance and controller load balancing rate to identify the set of under-loaded controllers in the SDN environment. It improves controller throughput, migration costs and response time by running different modules like load judgment, switch selection and controller selection. But it splits the load only into two intervals, i.e., overloaded and under-loaded, which limits its capability to distinguish the controller load status accurately. This will lead to low performance, uneven load balance and inaccurate load status. To solve these problems, this paper proposes MIBLB that modifies the ISMDA algorithm by implementing a multi-level load interval, i.e., idle, normal, overloaded and highly overloaded, and applies

dynamic threshold for switch migration according to the load balancing rate. The efficiency will be evaluated in terms of network performance parameters such as response time, throughput, and number of switch migrations.

1.2. Objectives

1.2.1. General Objectives

- The main objective of this research is to address the issues of ISMDA, i.e., uneven load balance and inaccurate load status, by providing an optimized efficient solution for multi-controller SDN load balancing.

1.2.2. Specific Objectives

- To modify the ISMDA algorithm to improve its load balancing efficiency
- Minimize network congestion and packet loss during the migration process.
- To ensure control plane has the optimal load balance rate after load balancing
- To do analysis on the performance of the ISMDA and MIBLB load balancing algorithms in terms of throughput, response time, and number of switch migration parameters
- To avoid unnecessary switch migration which will lead to communication overhead
- To improve performance by making accurate switch migration decisions

1.3. Scope and limitation of the study

1.3.1. Scope

Developing Multi-level interval based load balancing algorithm for multi-controller SDN involves addressing several problems such as improper division of load interval, determining the number of controllers and multi-controller architecture in the SDN, and balancing load in the controllers. However, this research will focus only on balancing the load on a given set of controllers. We also do not aim to determining the location of the controllers, which will have an impact in determining the load on the controllers.

1.3.2. Limitations

Different researches are conducted to address the load balance issue in SDN. This research, however, focuses on addressing the limitation of the state-of-the-art work, i.e., ISMDA, load balancing algorithm. For performance evaluation, the incoming load traffic is generated by iterating the load number at random. However, the iteration is set to 1500 times to make it more reliable. Additionally, the default threshold and maximum threshold are set to 2000p/s and 2200p/s respectively. But in a real scenario, the default threshold and maximum threshold could be set based on the controller's processing capability or its specifications.

1.4. Methodology

The research begins by collecting related papers and analyzing the load balancing of SDN controllers to identify the current relevant issues and problems in terms of load balancing. Then one state-of-the-art algorithm(ISMDA) is selected for more analysis, and the gap identification of that algorithm is performed to explore existing limitations. Subsequently, the problem statements and research objectives are defined. After that, the ISMDA algorithm will be modified and optimized to overcome its limitations. Then the proposed MIBLB load balancing algorithm will be implemented and its performance will be evaluated.

1.5. Contribution of the research

Due to the appearance of load balancing challenges in distributed SDN, there is currently a need for a multi-controller load balancing algorithm that takes into account performance, response time, resource allocation and communication overhead. Generally, the main contributions and novelties of this research work are as follows:

- Designing a multi-level load interval mechanism to select the controller that is overloaded in advance for the migration process.
- Proposing the MIBLB approach, an extended version of ISMDA, for effective switch migration to bring better network stability and load balance rate in the control plane.
- Implementing and evaluating the performance of the MIBLB algorithm against the baseline algorithm by considering throughput, response time, and number of migrations.

1.6. Thesis organization

The rest of this paper is organized as follows. Section II presents about software defined networking, overview of the ISMDA algorithm and the related work. Section III provides the proposed methodology which includes proposed system model, architectural diagram of proposed methodology, and flow chart and pseudocode algorithm of MIBLB. Section IV explains the simulation details of MIBLB, considering simulation environment, simulation results and discussion. Finally, Section V presents the conclusion and future work of the thesis.

2. Literature Review

2.1. Software Defined Networking

Software-Defined Networking is an evolutionary networking paradigm which has been adopted by large network and cloud providers. SDN is a vendor-independent network architecture developed by the Open Networking Foundation (ONF) as an initiative to address the requirements of next-generation networks. Network programmability [10] is a proposed solution to tackle network complexity challenges and help the internet be updated based on the emerging applications and services. Particularly, SDN decouple control plane from data plane which leads to considerably simplified network configuration and management along with enhanced flexibility and agility. The main idea behind SDN is to allow a logically centralized software-based controller (i.e. control plane) that takes care of network intelligence and decision making, while data plane is responsible for traffic forwarding tasks. The SDN is the new network framework which is programmable and vendor neutral [11].

2.1.1. Architecture of SDN

The architecture of SDN has three planes (data plane, control plane, and application plane) [12] as shown in Figure 1. The architectural layers of SDN are briefly described below.

A. Control plane

The control plane (controller) represents an abstract view of the complete network infrastructure, enabling the administrator to apply custom policies or protocols across the network hardware. For the operation, management, and maintenance of the entire network, a number of abstracted applications are operating in the controller. The SDN controller typically offers and maintains a comprehensive picture of the entire network from which users can build additional applications to improve network efficiency and resource utilization.

B. Northbound application interfaces

The northbound application programming interfaces (APIs) are the software interfaces between the SDN applications running on top of the network platform and the software modules of the controller platform. A northbound interface API is available to application developers via the network operating system (NOS). The low-level instruction sets needed to manage or monitor

forwarding devices are often abstracted by an API via the southbound interface.

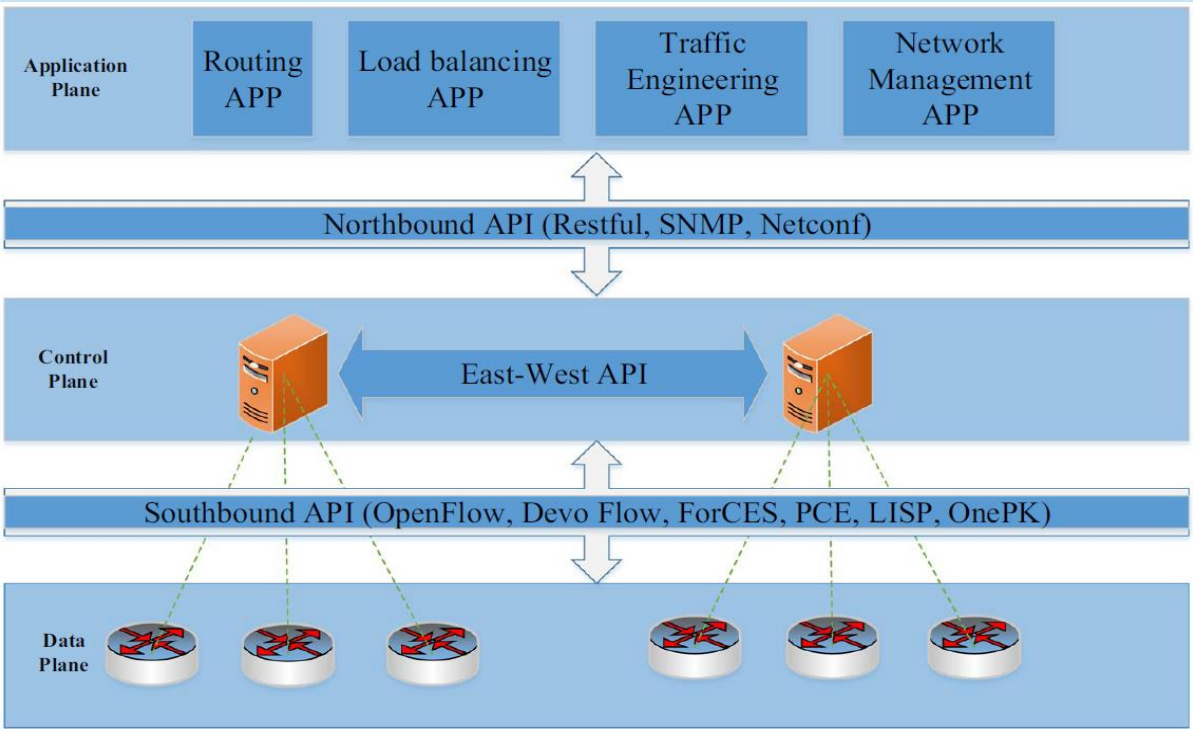

Figure 1 Overview of SDN architecture [13]

C. East–West protocols

The East-West interface protocol controls interactions among the various controllers in a multi-controller architecture.

D. Data plane and southbound protocols

In the SDN network architecture, the data plane stands in for the forwarding hardware. The controller needs the "southbound protocol," which is the OpenFlow protocol, in order to communicate with the network infrastructure [14]. The southbound interface API is used to specify the forwarding devices instruction set. The communication protocol between the components of the control plane and the forwarding devices is likewise defined by the southbound interface. The interaction between components of the control and data plane is formalized by this protocol.

2.1.2. OpenFlow protocol

The OpenFlow protocol is used to manage the generalized SDN architecture's southbound interface. The SDN architecture's first established standard interface for facilitating communication between the control and data planes. OpenFlow allows software-based access to flow tables, which advise switches and routers on how to route network traffic.

Administrators or controllers can quickly alter network configuration and traffic flow using these flow tables. The OpenFlow protocol also offers a fundamental set of administration tools that may be used to handle aspects like topology modifications and packet filtering [11].

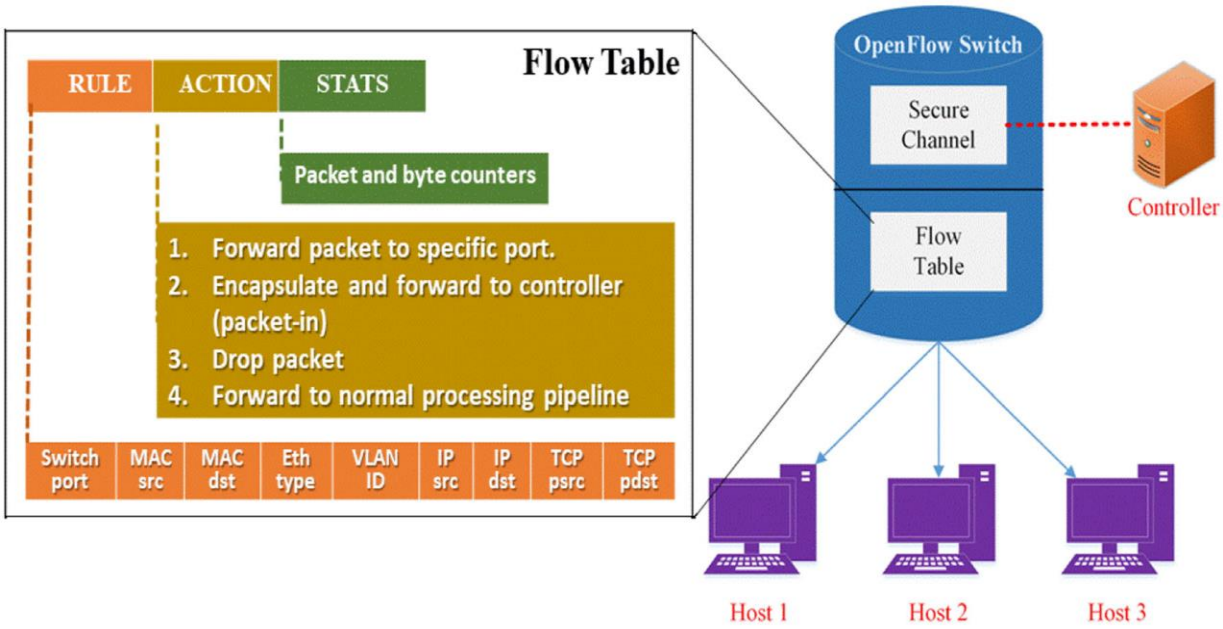

Figure 2 Basic architecture of OpenFlow [11]

2.1.3. Multi-controller deployment in SDN

A multi-controller is a set of controllers working together to achieve some level of performance and scalability. With the explosive growth of internet traffic and scale [15], the use of a single controller for network management makes the network less resilient and raises reliability issues. A single controller becomes a single point of failure, which is critical for network performance and reliability. Multiple controllers are typically implemented in networks to realize distributed control management in order to avoid the problem of low network performance and single point failure caused by an overburdened single controller. The control plane is divided into numerous sub-domains by multi-controller, and each controller is exclusively responsible for the switches in its own domain. As a result, it can alleviate the control plane's deficiencies in terms of reliability, scalability and versatility.

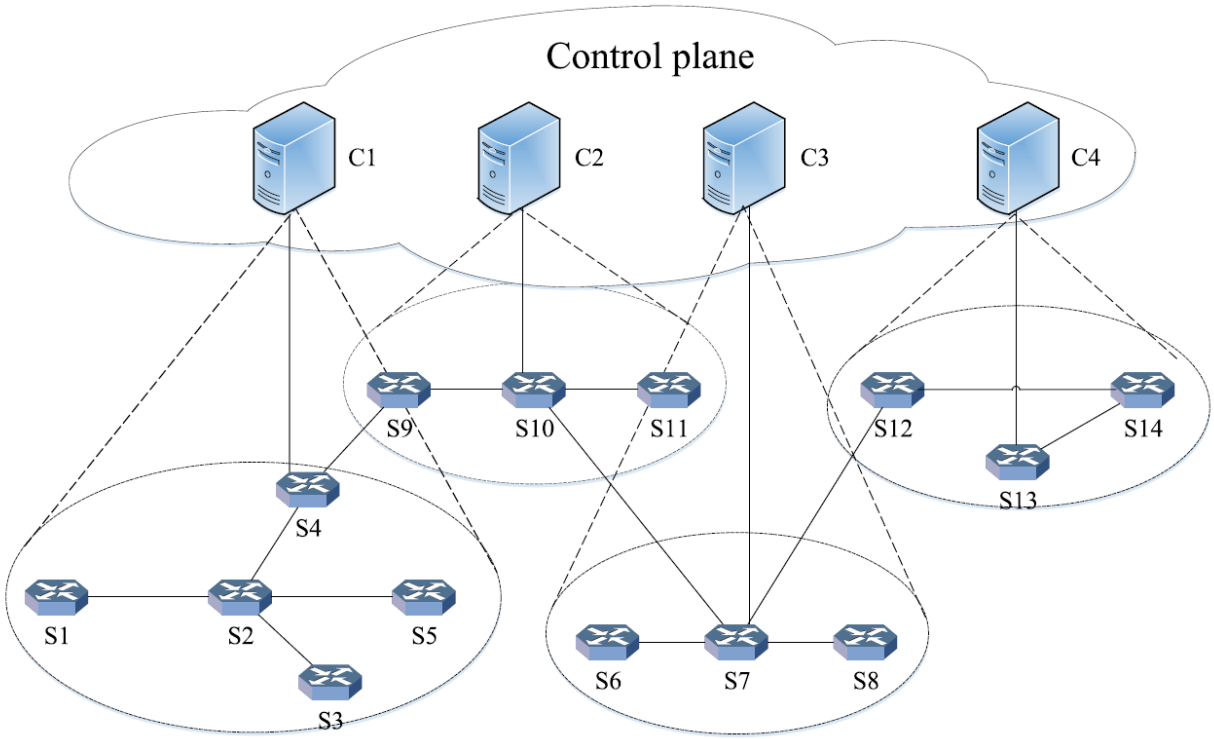

Figure 3 Multi-controller deployment in SDN [15]

2.1.4. Load balancing for Multi-controller in SDN

Load balancing is the process of dividing the workload into the individual controllers in distributed SDN in order to avoid overload [16]. Controller optimization methods and switch migration methods are two types of multi-controller load balancing techniques. To provide load balancing, the controller optimization approach uses an optimization algorithm to determine the optimal number and position of the controller on the network. The switch migration approach, on the other hand, is used to achieve controller load balance by migrating the overloaded controller's switch. The switch migration method is used in this study to complete the remapping of the relationship between the switch and the controller, which enhances network flexibility and dynamic load balancing. There are different types of load balancing algorithm of multi-controllers, some of them are explained below.

A. Reinforcement Learning LB mechanism (RL-LBM)

It designs a selection algorithm [17] in the phase of switch migration, which is used to select the out-migration domain, the in-migration domain and the switch to be migrating, as combined into one switch migration triplet. By using reinforcement learning, the load balancing mechanism chooses a set of switch migration queues that can achieve global optimization from local optimization.

B. Self-adaptive load balancing (SALB)

With multiple switch migration from source controllers to target controllers, this technique [18] dynamically balances load across several controllers. The key feature of this approach is an effective load distribution under high load conditions while taking the distance between switches and target controllers into account at the same time.

C. Efficient switch migration based load balancing (ESMLB)

This algorithm [19] effectively distributes switches to an underutilized controller. In this framework, a multi-criteria decision-making technique called Technique for Order Preference by Similarity to an Ideal Solution (TOPSIS) has been utilized to choose a target controller among a wide range of possibilities. This framework allows for flexible decision-making processes for selecting controllers with varying resource properties. It examines the control plane's real-time load information and determines whether or not to conduct switch migration.

D. Dynamic and Adaptive Load Balancing (DALB)

Without a single centralized component, DALB [20] is made for distributed architecture-based controller load balancing. It uses an adjustable load collection threshold to reduce the overhead of exchanging messages for load collection, and it can make policy and election localization decisions to reduce decision delay caused by network transmission. During switch migration, this algorithm chooses the controller of the nearest neighbor from a group of under-loaded controllers to accept load shifting.

E. Controller Adaption and Migration Decision (CAMD)

The CAMD [21] load balancing approach, which is based on switch migration, successfully chooses both the switch to be migrated and the target controller in a way that has little impact on response time. This effective switch migration technique offers better load balancing, faster response times, and higher end-user service quality. When the controller becomes overloaded, the CAMD calls the switch migration module while also identifying the under-loaded controller set based on the network's mean load.

2.2. Overview of an Improved Switch Migration Decision Algorithm

In order to guarantee that the most assembled resources are left available throughout the migration, ISMDA [6] is designed to select a severely overloaded switch to migrate from the overloaded controller. However, it will do so to the most suitable controller during the switch migration. When there is high traffic flow in the incoming load, the ISMDA approach is intended to address the network issue. During the controller load imbalance phase, the balancing component of the switch migration approach, which operates on each controller, is started. The ISMDA framework employed the controller variance and controller mean load status to identify the group of under-loaded controllers in the SDN environment. The most effective controller among the group of under-loaded controllers was chosen by simultaneously identifying the cost of migration and controller variance using the designed efficient migration model.

2.2.1. Load balancing mechanism for ISMDA

Load balancing through a switch migration technique is one of the efficient ways to manage an overloaded controller. This framework makes equal load distribution among controllers using an adaptive switch migration method to increase migration efficiency and improve load balancing rate. The system architecture of ISMDA is shown in Figure 4. The ISMDA is made up of three distinct modules that execute on each controller in the SDN network topology. When the controller's load capacity exceeds a predefined threshold, the first algorithm, the load judgment module is initiated immediately. The mean and variance of the controllers are calculated for each controller in the network by collecting all other controller loads, including its own. Then they identify the topology's group of under-loaded controllers, which is any group of controllers whose load weight is less than or equal to the mean or average of the controller's loads.

Algorithm: Representation of ISMDA algorithm

1: **Input**: N, β', ρ, *CLcur*

2: **Output**: ***Balanced multi-controller controllers***

3: **if** *CLcur* $>$ β **then**

4: *start load judgement module*

5: *start switch selection module*

6: *start controller selection module*

7: *Switch migration (cj ← sk)*

8: *Modify N (current and target)*

9: **end if**

10: return ***Balanced multi-controller controllers***

As a result, the switch selection module, as well as the load judgment module, are both triggered concurrently. The fundamental role of the switch selection module is to choose the highly loaded switch for migration purpose from the overloaded controller, which enhances selection effectiveness due to the good load balancing rate. Alternatively, the controller selection module, in addition to modules one and two, also initiated. It makes sure that the most suitable controller from the under-utilized controller group is chosen, which is completed with carefully considering migration efficiency and ensuring that the greatest number of clustered resources are available. The distributed controllers' mutual information (controller thresholds and load information) is saved in the distributed database (Hazelcast) storage application for easy access.

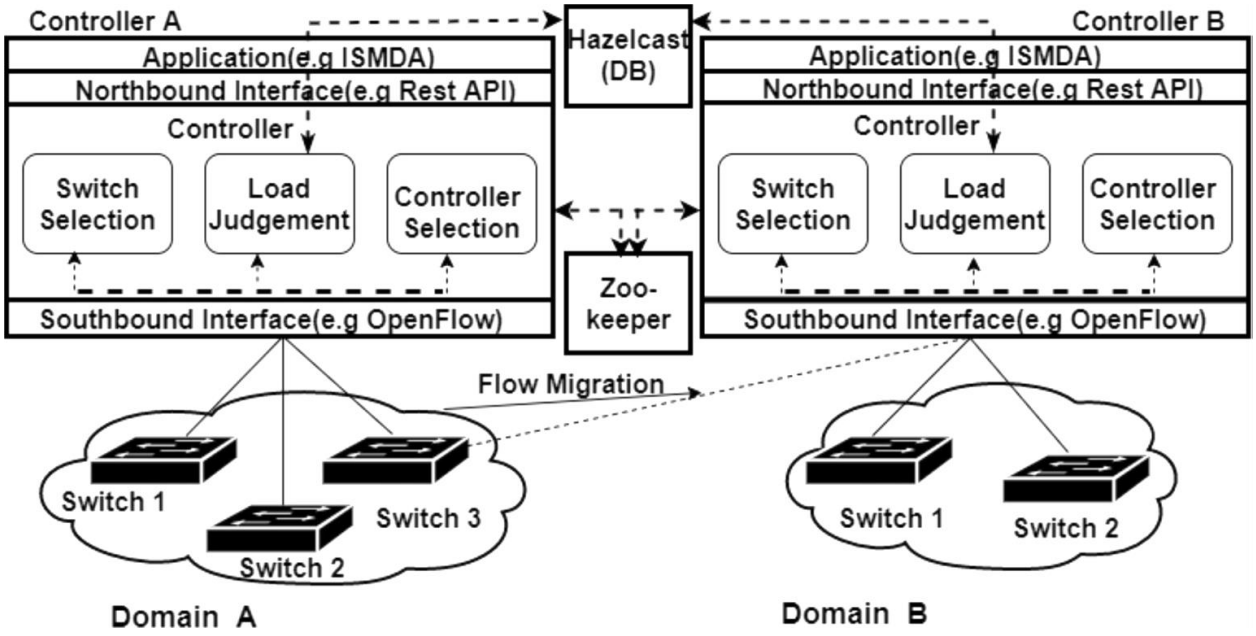

Figure 4 Load balancing mechanism for ISMDA strategy

2.2.2. Dynamic controller threshold

The dynamic threshold algorithm is used to adjust the controller threshold dynamically. The threshold value is saved in the distributed database. Each controller in the SDN environment checks to realize whether its load exceeds the threshold that has been predetermined. If the situation is met, it computes the average load of entirely networked controllers. It returns the

initial threshold if the calculated average load is less than or equal to the initial default threshold. If not, the calculated mean load is defined as the new threshold and adjust itself dynamically. To properly balance load, ISMDA must gather all other controller loads present in the system. However, with a static controller threshold technique, frequent gathering of load information messages in the SDN environment to create load balancing judgement can decrease system efficiency. The issue of decreasing system performance is solved using a dynamic load collection threshold technique. The controller threshold saved in the distributed database, where it is simply accessible to all controllers. But the state of the art (ISMDA) did not implement dynamic threshold in the experiment.

2.2.3. Load judgement module

The load statistics for each controller are monitored and reported by this module in real-time. The statistics encompass controller resources such as bandwidth utilization, CPU utilization, and memory utilization. When a controller's load exceeds a predefined threshold, it computes the mean load of entire controllers and employs a system variance (system performance error) to decide which controller is under-loaded. When the system performance error is low, the system performs better. As a result, any controllers whose current load is less than or equal to the system mean load are considered as under loaded. Each controller in the domain computes its overall CL(ci) load. Controllers are divided into two groups based on the statistics collected which is overloaded controllers and under-loaded controllers.

2.2.4. Switch selection module

This module chooses the highly loaded switch from the overloaded controller for migration process. The load judgment module provides useful statistics such as round trip duration, switch flow table entries, and packet arrival rate. While round trip time and flow table entries are normally employed for controller selection, the average message arrival rate is used for both controller selection and threshold estimate. As a result, when selecting a switch for migration, the load on the migrated switch should be equal to or less than the difference between half of the overloaded controller and the target controller.

$$SL_{Migrate} \leq \frac{\boldsymbol{CL}Overloaded - \mathrm{CL}Target}{2}$$

2.2.5. Controller selection module

The goal of this module is to choose the most appropriate controller capable of accepting the load from the overloaded controller in order to guarantee that the extreme clustered resources

are kept. A migration efficiency method is intended to classify both migration cost and load balancing variance concurrently in order to select the best controller. To further optimize their utilization on the efficiency model built, the migration cost and load balancing variation are defined. When switches are migrated, the network's load balance rate increases. However, it does increase the network's migration costs. The quantity of packets exchanged between controllers and the resulting rise in controller load are the two primary factors in migration costs. Migration cost of migration a switch s_k from controller c_i to c_j is defined as $mc_{skcj} = mc_{ex} + mc_{lc}$ where mc_{ex} is the message exchange cost and the mc_{lc} represent the increased load cost.

2.3. Related works

Numerous studies have concentrated on increasing SDN efficiency utilizing various protocols and techniques in order to address the load balancing problem. Load balancing using a switch migration technique is one of the effective schemes for resolving imbalance issues in SDN [22]. The authors conducted a thorough literature review on SDN load balancing in [8], [9] to address the difficulty of architectural designs and a taxonomy of current emergent load balancing strategies in SDN to improve service quality. They explore that multiple techniques did not consider some essential criteria that enhance the efficiency of the existing techniques. Furthermore, the authors in [23], [24] analyze the impact of SDN design and the architecture of its control plane on the switch migration load balancing methods efficiency. They stated that more research should be conducted to develop more secured and scalable controller in SDN control plane. Zhang, et al. [25] proposed an algorithm which minimize the mean response time of the control plane during each switch migration according to the real-time request distribution. The algorithm was utilized to enhance the load balancing rate indefinitely until no switches needed to be transferred. The efficiency of the entire network architecture will be reduced by frequent switch migration, which will raise the additional overhead of the network resources. To reduce the response time, Aly, et al. [26] suggested a model that accounts for the latency between the switches and the controllers that regulate them. Only the migration cost between the switch to be migrated and the target controller is taken into account by choosing the closest controller as the target controller. This resulted in the additional overload problem caused by load balancing switch migration.

Xing Ye, et al. [27] proposed a technique to reduce communication costs by migrating some switches onto light-loaded controllers so that all switches can be responded under the same performance during dynamic changes of network traffic. However, the method requires a

significant number of experiments to obtain parameters that ensure its convergence to the optimal. Lan, et al. [28] proposed a mechanism that can dynamically shift the load from the heavily loaded controller to the lightly loaded one via switch migration based on load status on each controller. However, it is applicable only to a hierarchical architecture topology.

For the purpose of resolving the controller load balancing issue using the switch migration technique, numerous research have been carried out. High delay or latency is however produced while relocating switches when the controllers are overloaded [29]. To solve the problem of latency, H. Zhong, et al. [30] proposed the Assessing Profit Of Prediction (APOP) scheme, a load balancing approach in the multi-controllers based on the overloaded state prediction and profit assessment. But the author considers only the latency metrics. During migration of OpenFlow(OF) device, there is the increasing of the response time, jitter and packet loss. To improve the study by [30], Raj Kumar, et al. [31] proposed a method in which the response time, jitter and packet loss are minimized during device migration by using *liveness, safety*, and *serializability*. However, the controller role is based on master-slave method in which master controller failure cause single point of failure.

Yang, et al. [5] introduced a multi-controller dynamic load balancing method based on reinforcement learning to solve the problem of switch migration conflict. In each domain, the switch in the overloaded controller is migrated to the lighter controller by adopting a request response model integrated with an improved state action reward state action (SARSA) reinforcement learning algorithm. But there is high complexity of the algorithm and also QoS is not considered. A supervisor controller-equipped framework with Laman was suggested by Nayyer, et al. [32] with the purpose of preserving the centralized architecture approach and enhancing SDN scalability. A central controller known as the Supervisor Controller serves as the hub for all other controllers in the framework. By spreading out the bulk of the supervisor controller's incoming load, Laman can lighten the burden on the centralized controller, which in turn enhances scalability. However, having a single supervisor controller in charge of all the network's controllers could result in a single point of failure.

Based on reinforcement learning, Zhuo Li, et al. [17] designed an algorithm for the switch migration phase. The load balancing technique employs reinforcement learning to select a collection of switch migration queues capable of achieving global optimization from local optimization. The mechanism can make maximum use of the controllers' resources, quickly balance the load, eliminate migration overhead, and achieve a faster packet-in request response

rate. However, the migration mechanism is based on average load rate only, other parameters such as packet loss and migration cost are not considered. To tackle the switch migration problem, N. Correia, et al. [33] proposed a heuristic technique with robust solution based on a local search algorithm that takes into account shift and swap motions. A search strategy incorporates shift and swap moves. Every action is assessed based on how much advantage it will provide to the controllers of immigration and outmigration. but large variations in load distribution may result in frequent migrations, lowering control plane performance. Furthermore, K. S. Sahoo, et al. [19] provided a framework for decision-making about controller adaptation and switch migration that can assess the load in real-time to decide whether to migrate switches. This framework offers adaptable selection procedures for controllers with various resource properties. The introduction of multi-criteria decision technology to choose the target controller resulted in a reduced level of efficiency without taking the cost of migration into account.

Sahoo, et al. [21] presented the Controller Adaption and Migration Decision (CAMD) model, which is based on the least-loaded controller selection from a group of under-loaded controllers for receiving load shifting while switch migration. This model is effective and capable of supporting higher loads when the incoming traffic is of moderate flow (traffic load less than 500 p/s). However, choosing a switch with the minimum load for migration would result in a low load balancing rate. Similar results are obtained from the CAMD system, including decreased network efficiency, limited throughput, high response times and substantial packet losses. To improve the CAMD algorithm, Adeyoka, et al. [6] adopted an Improved Switch Migration Decision Algorithm (ISMDA) to handle the network difficulty of elephant flow (traffic load more than 500 p/s). The average load status and controller variance were employed by this framework to identify the network's under-loaded controllers. The ISMDA technique centered on efficiently migrating highly loaded switches from one overloaded controller to the best-suited controller among an under-loaded collection of controllers. However, because the algorithm is unable to precisely determine the controllers' load status, resulting in a reduction in migration efficiency. C. Min, et al. [34] combined the advantage of the ant colony algorithm and genetic algorithm fusion, which is designed to optimize the load balancing problem of SDN under the control plane through the ant colony algorithm. The combination of the algorithms, however, has resulted in a relatively high complexity.

Generally, the reviewed papers proposed different load balancing approach for multi-controllers SDN environment. Some of these algorithms are specific to SDN in datacenter, IoT, cloud computing and 5G mobile network [8], [9]. To evaluate the performance of the designed algorithm, different type of simulation tools such as Floodlight, OpenDayLight(ODL), SDN POX are used [33], [23], [29]. They have also used many techniques to balance load in the network depending on their specific target benefit. For this research, I preferred to modify the ISMDA algorithm because it is applicable in elephant flow traffic (high traffic) and large-scale network. This study modifies the state of the art (ISMDA) algorithm to improve the performance and inaccurate load balance issue in the distributed controllers.

2.4. Summary of Related Works

Reference	Type of algorithm	Method of work	Advantages	Limitation
Zhuo Li, et al. [17] (2021)	Reinforcement Learning LB mechanism (RL-LBM)	In the switch migration decision phase, the algorithm selects a sequence of switch migration triples (the migrate-out domain, the migrate-in domain, and the migrating switch), then creates an ideal migration model based on reinforcement learning.	The approach can fully utilize the controllers' resources, quickly balance the load between controllers, avoid unnecessary migration overhead, and achieve a faster packet-in request response rate.	The migration mechanism is based on average load rate only, other parameter is not considered.
N. Correia, et al. [33] (2019)	Migration competency-based load balancing (MCBLB)	It employs a local search framework, MCBLB can begin with an infeasible solution, employs several evaluation functions (one for each type of move), and filters the potential movements. Workload information should be examined on a regular basis and compared to a threshold to detect over utilized controllers.	Selecting a group of switches to migrate can eliminate the problem of load oscillation while also providing a timely efficient solution and greatly reducing the number of decisions necessary.	The migration of a switch does not restore the load to its usual condition. Large variations in load distribution may necessitate frequent migrations, which degrades the performance of flow setup.
Shike Yang, et al. [5] (2020)	Multiple controller dynamic load balancing method based on intelligent	It uses the SARSA(state action reward state action) reinforcement learning framework, combined with switch migration prediction method for migrating the switch in the overload	It reduces the communication overhead between controllers and improve the system throughput	QoS is not considered Increased complexity

	collabora-tion	controller to the lighter load controller. □ Multiple controllers that are close to each other are divided into the same area		
Sahoo, et al. [21] (2019)	Controller adaption and migration decision (CAMD)	□ Uses a load-balancing strategy based on switch migration that selects both the switch and the destination controller with the least amount of reaction time. □ During overloading, it uses the network's mean load to identify the unloaded controller set and simultaneously calls the switch migration module.	□ This method is efficient and accepts greater loads when the incoming traffic is continuous. □ It outperforms the least loaded controller in terms of throughput, reaction time, and migration cost.	The CAMD technique results in decreased network efficiency, low Load Balancing Rate (LBR), and high packet loss.
Congco ng Min, Et al. [34] (2021)	Improved ant colony algorithm	The advantages of the genetic algorithm and ant colony algorithm are integrated, which improves the performance of SDN to a certain extent.	□ The advantages of both algorithms are fully utilized to greatly improve the efficiency of SDN load optimization. □ The dual purposes of migration and load balancing between multiple controllers are finally achieved.	□ Relatively high complexity of the algorithm. □ The advantage is visible only when the load of the controllers are high.
S. Sahoo et al. [19] (2020)	Efficient switch migration based load balancing (ESMLB)	For achieving a better load balancing, the proposed ESMLB monitors the real-time load information of the control plane and determines whether to implement switch movement or not.	□ A multi-criteria decision technique TOPSIS has introduced for choosing the target controller. □ Following the selection process, the selected switch shifts to the target controller domain.	The scheme does not consider the migration cost.

Table 1 Summary of related works

3. Proposed Methodology

This section presents the proposed methodology, Multi-level Interval Based Load Balancing (MIBLB) algorithm. It covers the proposed system model, architectural diagram and flow chart of the MIBLB algorithm.

3.1. Proposed system model

The proposed MIBLB model is used to identify both load balancing mean and load balancing rate for the best controller selection from the group of idle and normal controllers. The goal of this model is to identify a suitable algorithm that mainly considers data load flow arrival at each SDN switch and relates it to the computational load of each SDN controller. The model then formalizes the challenge of controller load balancing into an optimized algorithm. An SDN, S, is composed of a set of k switches $\mathcal{S} = \{s_1, s_2, \ldots, s_k\}$, $S_k \in S$, controlled by a set of n controllers $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$, $C_n \in C$. This study makes the premise that each controller has been put appropriately in the topology so that it can individually control a group of switches. The MIBLB algorithm divides the load into four gradual load levels which are idle, normal, overloaded, and highly overloaded intervals based on the quarter method. The MIBLB method consists of three modules. The module functionalities are defined below.

3.1.1. Load judgement module

This module estimates the average load while tracking and reporting the current controller load. The load judgement module identifies idle and normal controllers for load shifting. The report statistics cover controller resources including bandwidth, CPU, and memory utilization. Each domain controller calculates its overall load, CL(ci). This can be accomplished in real-time by employing the controller resources metrics listed above, along with their respective allocated weights, which is expressed as:

$$CL\,(ci) = \begin{bmatrix} w1 & w2 & w3 \end{bmatrix} \begin{vmatrix} CL_{band}(ci) \\ CL_{mem}(ci) \\ CL_{cpu}(ci) \end{vmatrix} \qquad (1)$$

where $w1$, $w2$ and $w3$ represents the weights of controller resources (bandwidth, memory and CPU) and *CLmem(ci), CLband(ci) and CLcpu(ci)* denotes the real memory, bandwidth and CPU used on the control plane by switch data flow request. Since all the controllers are taken

into account, the population variance idea is added in order to determine the control plane equilibrium level. This variance is stated as follows:

$$\lambda = \frac{1}{n} \sum_{i=1}^{n} (\mathrm{CL}(ci) - \overline{CL})^2 \qquad (2)$$

where λ represent the variance, n represents the number of controllers, CL(ci) denotes the controller load in real-time and $\overline{CL}$ indicates average load of controllers. As a result, the load balancing module will launch a load balancing module based on the variance. In addition, the switch migration will be initiated if the controller load imbalance (incoming load exceeding the default threshold) is detected. The module will gather the loads of the controllers in the network and calculate the average result of the load factor, which can be stated as:

$$\overline{CL} = \frac{1}{n} \sum_{i=1}^{n} CL(ci) \qquad (3)$$

The proportion of the controller's current load status to their mean load is expressed as a controller load balancing rate which is represented by ρ_i. Equation 4 is used to calculate the load balancing rate.

$$\rho_i = \frac{CL(ci)}{\frac{1}{n} \sum_{i=1}^{n} CL(ci)} \qquad (4)$$

The module then calculates the five categories listed below as the foundation for its migration decision-making:

- $\boldsymbol{\rho i \leq 0.5}$**:** This indicates that the current load of the controller is less than the mean and normal load. This indicates that the controller is idle and can accept extra loads or switch migration. The idle controller will have the highest priority for receiving switches.
- $\boldsymbol{0.5 < \rho i < 1}$**:** This shows the current controller load is normal and still have the ability to handle some unexpected situations.
- $\boldsymbol{1 < \rho i \leq 1.5}$**:** This shows that the current controller load is greater than the mean of the load. This indicates that the controller is overloaded, which would result in network congestion and load imbalance.
- $\boldsymbol{\rho i > 1.5}$**:** This shows that the controller is in an abnormal functioning condition and is highly overloaded, although it can still function for a short period of time. The switch is migrated if other controllers in the network are still in the idle or normal states.
- $\boldsymbol{\rho i = 1}$**:** This shows that the mean of all working controllers is equal to the current controller load status. As a result, the controller is neither overloaded nor under-loaded. As a result, the controller will not take on any more load.

Controllers are classified into a set of idle, normal, overloaded or highly overloaded based on the conditions listed above. This is mathematically represented as:

$$\begin{bmatrix} \boldsymbol{\rho i} > \mathbf{1.5} & \text{Controller in highly overloaded mode} \\ \mathbf{1} \leq \boldsymbol{\rho i} \leq \mathbf{1.5} & \text{Controller in overloaded mode} \\ \mathbf{0.5} < \boldsymbol{\rho i} < \mathbf{1} & \text{Controller in normal mode} \\ \boldsymbol{\rho i} \leq \mathbf{0.5} & \text{Controller in idle mode} \end{bmatrix}$$

3.1.2. Switch selection module

For the purpose of migration, this module chooses a switch which has the high load from the overloaded controller. According to the OpenFlow protocol [11], round trip duration, switch flow table entries and packet arrival rate are the helpful statistics used for the switch selection and load judgment module. The switch can forward a variety of messages to the controller, including hello messages, echo messages, and packet-in messages. However, packet-in flow messages add more to the controller's weight [11]. The average packet-in message forwarded from the switch to the controller was used as a measure of controller load in this study. As a result, when selecting a switch to be migrated, the weight on the migrated switch should be equal to or less than the difference between half of the overloaded controller and the destination controller. Hence, the switch is chosen using Equation 5.

$$SL_{Migrate} \leq \frac{\boldsymbol{CL}overloaded - \mathbf{CL}target}{2} \qquad (5)$$

$SL_{Migrate}$ denotes the load of the switch to be migrated, $CL_{overloaded}$ indicates the overloaded controller load, and the CL_{target} indicates the target controller load. But if the controller is highly overloaded, this equation will not be considered. Instead, highly loaded switch will be migrated to lightly loaded target controller to reduce the chance of packet loss and bottleneck.

3.1.3. Controller selection module

This module's main objective is to choose the most suited controller to receive the load from the overloaded controller in order to maintain load balancing in SDN. In this study, we developed a migration efficiency model that concurrently determine migration cost and load balancing variance at the same time for optimal selection of the best controller. When switches are migrated, the network's load balance improves. However, it will incur additional migration costs for the network. As a result, the idea of migration cost must be introduced. Migration costs are generally comprised of two major components, namely an increase in controller load and the number of packets transferred between controllers.

Equation 6 describes the migration cost($\mathbf{mc_{skcj}}$) during the migration of s_k from c_i to c_j, which is expressed as:

$$mc_{skcj} = mc_{ex} + mc_{lc} \quad (6)$$

where mc_{ex} represents the message exchange cost and the mc_{lc} represents the increased load cost. The increased load cost is the rise in load that occurs when the target controller receives the switch to be migrated from the controller that is overloaded. As a result, it is necessary to introduce the load variance technique as a selection factor. In this study, the load balancing factor used is the controller load variance, and $\overline{CL}$ as the average controller load of N controllers. This network selection factor prior to switch migration is reformulated from Equation 2 as indicated in Equation 7.

$$\vartheta = \frac{1}{n} \sum_{i=1}^{n} (CL(ci) - \overline{CL})^2 \quad (7)$$

where ϑ represent the variance in network load before switch migration. CL(ci) indicates controller's load and $\overline{CL}$ represents average load of controllers in the network domain. The network's load selection factor following switch migration guarantees that system instability is kept to a minimal. This is stated as

$$\vartheta^* = \frac{1}{n}[\sum_{i=1}^{n} (CL^*(ci) - \overline{CL}^*)^2 + (CL^*(cj) - \overline{CL}^*)^2 \quad (8)$$

where $CL^*(ci) = (CL(ci) - \theta_{sk} mc_{skci})$ and $CL^*(cj) = CL(cj) + \theta_{sk} mc_{skcj}$

In Equation 8, ϑ^* represents the variance of controller load after switch migration has happened and θ_{sk} represents the packet-in messages forwarded from switch s_k. To further improve the choice of the target controller and to demonstrate adjustment between load balancing rate and migration cost, migration efficiency is generated. The proportion of load balancing rate to migration cost is used in this study to define migration efficiency. It is written as the equation below.

$$\Delta = \frac{|\vartheta* - \vartheta|}{\mathbf{mc_{skcj}}} \quad (9)$$

The use of Equation 9 depends on the status of controller load. If the current controller is highly overloaded or the target controller is idle, migration efficiency is not computed in order to reduce computation cost and packet loss.

3.1.4. Dynamic controller threshold

The dynamic modification of the controller threshold is carried out using this algorithm. Every controller in the network keeps track of whether the load has above a predetermined threshold. If the condition is met, it computes the average load of entire networked controllers. If the calculated mean is less than or equal to the predetermined threshold, the default threshold is returned. If not, a new threshold is set and dynamically updated using the anticipated average load. All other controller loads in the system must be gathered by MIBLB in order to correctly balance the controllers' load.

3.2. Architectural diagram of the MIBLB

The architecture of the MIBLB approach comprises three different modules, these are load judgement module, controller selection module and switch selection module. Load judgement module tracks the incoming controller load and estimate the average load and load balancing rate. Then, it identifies controller status in the SDN environment and categorize them into the set of *idle*, *normal*, *overloaded*, and *highly overloaded* controller for load shifting. The controller selection and switch selection module is used to select appropriate controller and switch during load imbalance respectively to shift the highly loaded switch to lightly loaded controller. The efficiency of the model is evaluated using performance evaluation metrics like throughput, response time and number of migration.

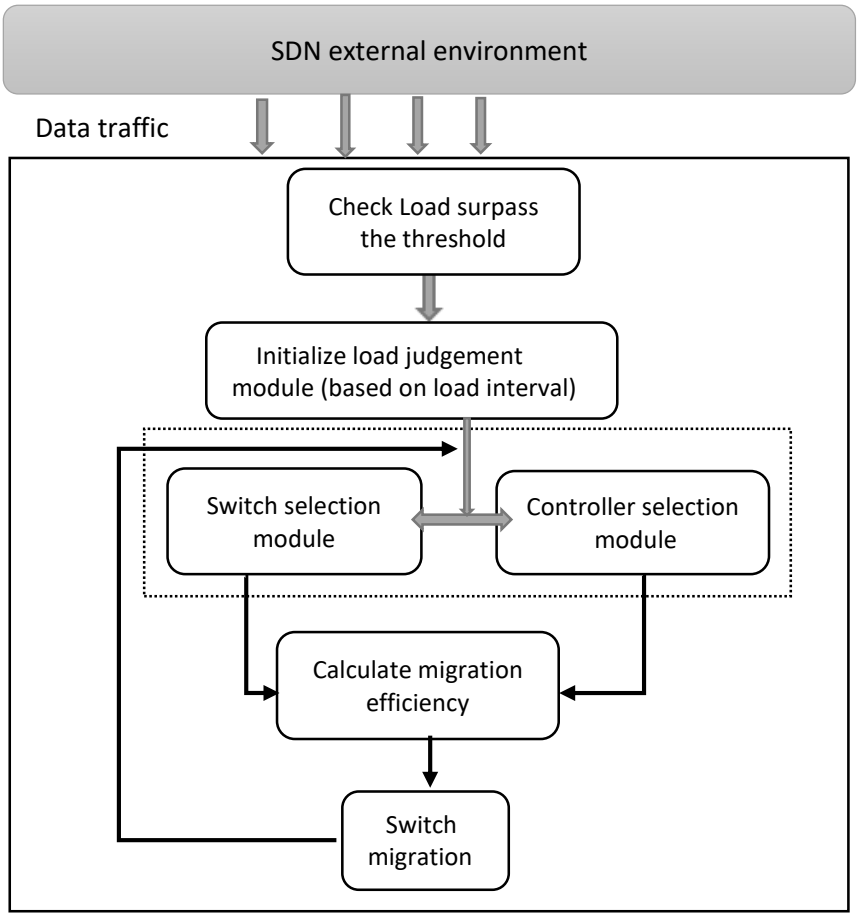

Figure 5 Architectural diagram of proposed methodology

3.3. Flow chart of MIBLB algorithm

Notation	Description
$C = n$	Controllers set (n number of controllers)
$S = k$	Switch sets (k number of switches)
β'	Default threshold
β_{max}	Maximum threshold
ρ	The load balancing rate
CL_{cur}	Current controller load
(CI)	The group of idle controllers
(CN)	The group of normal controllers
(COL)	The group of over-loaded controllers
(CHL)	The group of highly overloaded controllers
CI,CN, CHL and COL ∈ C	All sub-controllers are member of C
i^{th}	Controller that is a member of CI

j^{th}	Controller that is a member of CN
k^{th}	Controller that is a member of COL
l^{th}	Controller that is a member of CHL
P(*si*)	Group of switches managed by CI
P(*cj*)	Group of switches managed by CN
P(*sk*)	Group of switches managed by COL
P(*sl*)	Group of switches managed by CHL
Mc_{ij}	The mapping relationship between *sk* and *cj*

Table 2 Notations used in the MIBLB algorithm

Figure 6 depicts the various modules of MIBLB and their primary functions. When a controller's current load status exceeds the default threshold (β), the algorithm instantly invokes the load judgement module to determine the load status of the entire controllers in the SDN environment. The current overloaded controller (ρ) and the under-loaded controller(ρi) controller load balancing rate are then determined.

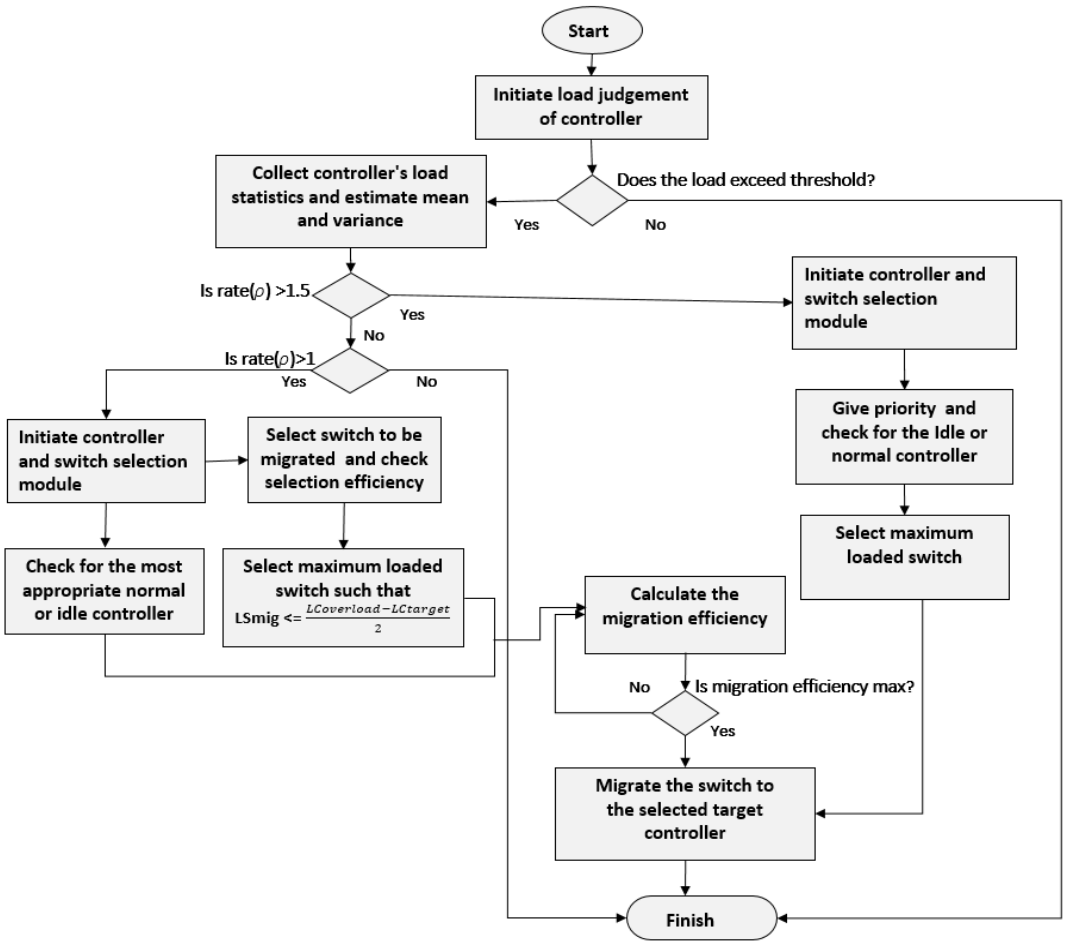

Figure 6 flowchart of the MIBLB algorithm

Based on the calculated rates the controller is categorized to the sets of idle, normal, overloaded and highly overloaded state by quarter method and then the efficient switch migration process is decided. Additionally, the modules *switch selection* and target *controller selection* are simultaneously called for switch and controller selection respectively. As a result, the algorithm makes a judgment and the selected switch migrates to the target controller.

3.4. Proposed pseudocode algorithm

To balance the controller load efficiently, Multi-level Interval Based Load Balancing (MIBLB) algorithm is proposed. Below are the details of the MIBLB's modules which include the load judgement module, controller selection module, switch selection module as well as dynamic threshold algorithm to achieve the optimal switch migration.

A. Algorithm for load judgement module

This module collects other loads in the SDN environment and calculates the average value of the load factor ($\overline{CL}$) and controller load balancing rate(ρi) which is used to indicate the ratio of the current load of the controller to their average controller load. By using the calculated load balancing rate, the load status of the controller is divided into four categories and based on these categories the priority is given to highly overloaded controller to avoid packet loss and bottleneck.

Algorithm 1: Load judgement module of MIBLB

1: Input: n, ρi, β'

2: Output: *CI , CN , COL, CHL*

3: for $i = 1$ *to* n do

4: *check controller load CL(ci)*

5: end for

6: *Calculate mean load* $\overline{CL} = \frac{1}{n} \sum_{i=1}^{n} CL(ci)$

7: for $i = 1$ *to* n do

8: $\rho i = \frac{CL(ci)}{\frac{1}{n} \sum_{i=1}^{n} CL(ci)}$

9: if $\rho i \leq 0.5$ then

10: $CI \leftarrow ci$

11: else if $\rho i > 0.5$ and $\rho i < 1$ then

12: *CN* ← *ci*

13: else if *ρi* ≥ *1* and *ρi* ≤ 1.5 then

14: *COL* ← *ci*

15: else if *ρi* > 1.5 then

16: *CHL* ← *ci*

17: end if

18: end for

19: return *CI, CN, COL, CHL*

B. Algorithm for Switch selection module

For the purpose of migration, this module chooses the highly loaded switch from the overloaded controller.

Algorithm 2: Switch selection module of MIBLB

1: Input: *COL, CHL*

2: Output: group of switches to be migrated *sk* from *CHL or COL*

3: *initialize switch set P* = {}

4: for ∀ *sk* ∈ *COL* or *CHL* do

5: for ∀ *Flows* ∈ *COL* or *CHL* do

6: *received_packets* ← *sk.flowstats()*

7: end for

8: end for

9: *calculate load balancing rate using* $\left(\frac{1}{n}\ \sum_{i=1}^{n} CL(ci)\right) / \max_{1 \le i < n} (Ci)$

10: *select switch to be migrated sk* = argmax{Psk} *sk* ∈ *COL* or *CHL*

11: *check switch(sk) to be migrated using SLmigrate* ≤ $\frac{CLoverloaded - \mathrm{CL}target}{2}$

12: return *switch set P*{}

C. Algorithm for controller selection module

The most appropriate controller to take on the load from the overloaded controller is chosen using this module.

Algorithm 3: Controller selection module of MIBLB

1: Input: *sk*, *CI, CN*, mc_{lc}

2: Output: target controller *cj*

3: *read current load on CI, CN*

4: for ∀ *controller cj* ∈ *CI* or *CN* do

5: for *j* = 1 *to n* ∈ *CI* or *CN* do

6: *calculate migration efficiency using equation (9)*

7: end for

8. end for

8: *check that CL(ci) + SL(si) ≤ CL(cj)*

9: *cj = argmax{ CI or CN }cj ∈ CI(Δ) or CN(Δ) : CL(ci) + SL(si) ≤ CL(cj)*

10: update { CI *or* CN } *with current load status*

11: return *cj*

D. Algorithm for MIBLB dynamic controller threshold

The dynamic modification of the controller threshold is carried out by this dynamic threshold algorithm. The algorithm will change the default threshold if the mean of the controllers in the network is greater than the predefined threshold and the updated threshold should also be less than the defined maximum threshold.

Algorithm 4: MIBLB dynamic threshold

1: Input: Controller's load list {*CL1*, ..., *CLn*}

2: Output: Adjustable (*ϐ'*)

3: *find mean load* $\overline{CL} = \frac{1}{n} \sum_{i=1}^{n} CL(ci)$

4: let Initial (*ϐ'*) = 2000 and $ϐ_{max}$ = 2200

5: if $\overline{CL}$ ≤ Initial(*ϐ'*) then

6: (*ϐ'*) = 2000

7: else if $\overline{CL}$ ≥ Initial(*ϐ'*) and $\overline{CL}$ < $ϐ_{max}$ then

8: (*ϐ'*) = $\overline{CL}$

9: end if

11: return (*ϐ'*)

4. Simulation

This section presents the simulation conducted with the proposed model and the findings. In the first part of this section, the simulation environment specification and the experimental parameters will be presented. Then the simulation results and analysis of the simulation will be explored. In the result analysis, the performance of the proposed algorithm is compared against the baseline, ISDMA and two other state-of-the-art works, DALB and CAMD. Finally, discussion and comparative analysis will be presented.

4.1. Simulation environment and parameter settings

For the performance evaluation in this study, a simulation technique and assessments of throughput, number of migration, and response time were used. The simulation is conducted on a laptop with an Intel Core i7-8665U CPU @ 4.01 GHz, 8GB memory and windows 10 enterprise OS. The proposed MIBLB load balancing algorithm is implemented on Jupyter notebook compiler with Python programming language for evaluation. The simulation is performed by varying the number of controllers and the incoming load. The iteration is set to 1500 in order to get accurate result. The load balancing efficiency of the proposed method was considered when the number of controller is two, four, and eight. To create a high load on the controllers, many packets are generated and it will be distributed to the appropriate under-loaded controller as per the programmed MIBLB load balancing algorithm. When the average load of the controllers (mean) is beyond the default threshold the threshold will be set to the mean dynamically. Hence, the packet will not be lost until the mean of controller's load reach maximum threshold. This simulation will run for multiple times (1500 times) by varying the incoming load and number of controller (two, four, and eight controllers) to verify the accuracy of performance evaluation in load balancing rate.

Parameters	Values
Number of controllers	2, 4, and 8
Default threshold	2000 p/s
Maximum threshold	2200 p/s
Current load	500 p/s
Low incoming load	10000 – 20000 p/s
High incoming load	21000 – 30000 p/s
Iteration	1500

Table 3 Summary of parameters used for the experiment

4.2. Performance evaluation metrics

To evaluate the proposed approach, we have used three evaluation metrics: number of switch migration, throughput and response time. Below we explain each metrics.

A. Number of switch migration

It is the number of migrations performed by each algorithm when a controller is overloaded. The number of migrations should be kept to a minimum in order to achieve optimal communication performance. Switch migration is the primary way for solving the load balancing problem when using several controllers, although too much switch migration might degrade the overall network service quality.

B. Throughput

The throughput of a controller, which measures how many requests it can handle in a given amount of time, which shows the controller's bearing capacity of controller. This statistic calculates the total number of tasks finished in a unit of time following load balancing. Using a load balancing strategy, it establishes the rate at which the computational job is completed. Throughput measures how many successful requests are handled from source to destination in a network in a given amount of time. It is measured in bps (bits per second). If more data is transferred, the controller will have higher throughput result. Throughput, the transmission of data calculated in a particular time duration, is given by:

$$\text{Throughput} = \frac{\text{Transmitted data byte}}{\text{Time taken for transmission}}$$

C. Response time

The time taken by the controller to respond to the respective client requests measure the response time. It is described as the length of time it takes for a user to obtain a request's results. Response time measured in milliseconds (*ms*) is calculated based on the time difference between the request for service from the switch and the response time for the incoming request from controller. The controller response time is an indicator that can be used to reflect the state of the controllers in the distributed controllers. The better the controller status, the faster the response time. When there is a load imbalance, the controller's response time increases dramatically. Bandwidth, the number of users using the network at the same time, the amount of requests, and the average processing time are some of the different factors that affect response time.

4.3. Simulation results and discussion

The MIBLB algorithm is developed to balance the load among multiple controllers in a multi-controller SDN paradigm. MIBLB takes into consideration different metrics such as throughput, response time, and number of migration for evaluation in order to improve the overall performance the SDN network. To have an efficient multi-controller load balancing method based on switch migration, the throughput should be high and the response time and number of switch migrations should be very low. This study conducts the simulation by varying the input parameters such as incoming load and number of controllers to check the performance of the proposed algorithm. To validate the performance of the MIBLB algorithm, a series of simulation experiments were set up, and the MIBLB was contrasted with ISMDA and two other state-of-the-art works, DALB and CAMD. DALB [20], known as the nearest controller selection model, simply causes traffic bottleneck when accepting load shifting. It raises the communication overheads, since numerous switches may move into the closest controller at one occasion. CAMD [21], or least-loaded controller selection, can be effective in accepting load shifting only when the incoming data traffic is low. CAMD's selection of a switch with the least flow request rate for migration is insufficient to maintain load balance when contrasted to a severely loaded switch. Choosing a switch that is lightly loaded may not enhance network efficiency as much as picking a switch that is highly loaded. For migration purpose, ISMDA [6] chooses a maximum loaded switch. It analyzes the controller variance and average load status to identify the network's under-loaded controllers. But ISMDA is not designed in the way that it can distinguishing the controller load status accurately. MIBLB algorithm is an extended version of ISMDA which is based on dividing controller load interval into four specific group to accurately categorize controller load status. It also applies dynamic threshold to avoid network bottleneck and packet loss. The maximum threshold should be set depending on the controller computing capacity. For migration, the suggested MIBLB algorithm chooses a switch that is highly loaded. It gives priority to highly overloaded controller and carefully chooses the most suitable controller by collecting accurate load status to accept the load. This section discusses the simulation's results and provides a brief explanation of each performance evaluation metrics.

4.3.1. Throughput

On the basis of the traffic produced throughout the simulation procedure, the average controller throughput was estimated. In the experiment, the incoming load and number of controller's parameters is varied to observe the change of controller's throughput. Since the load starting from 1 p/s to 9000 p/s does not cause load imbalance on the controllers, this range of load is not included in the experiment. The total incoming load is ranged from 10,000p/s to 20,000p/s for low load and 21,000p/s to 30,000p/s for high load. Figure 7 shows traffic flow distribution of the incoming load. These loads are generated from different switches randomly and flow to their respective domain controllers. The MIBLB algorithm's efficiency is unaffected by traffic generation type, whether it is exponential or random, because the traffic range in the experiment is set to low or high for incoming load traffic based on the controller's capability.

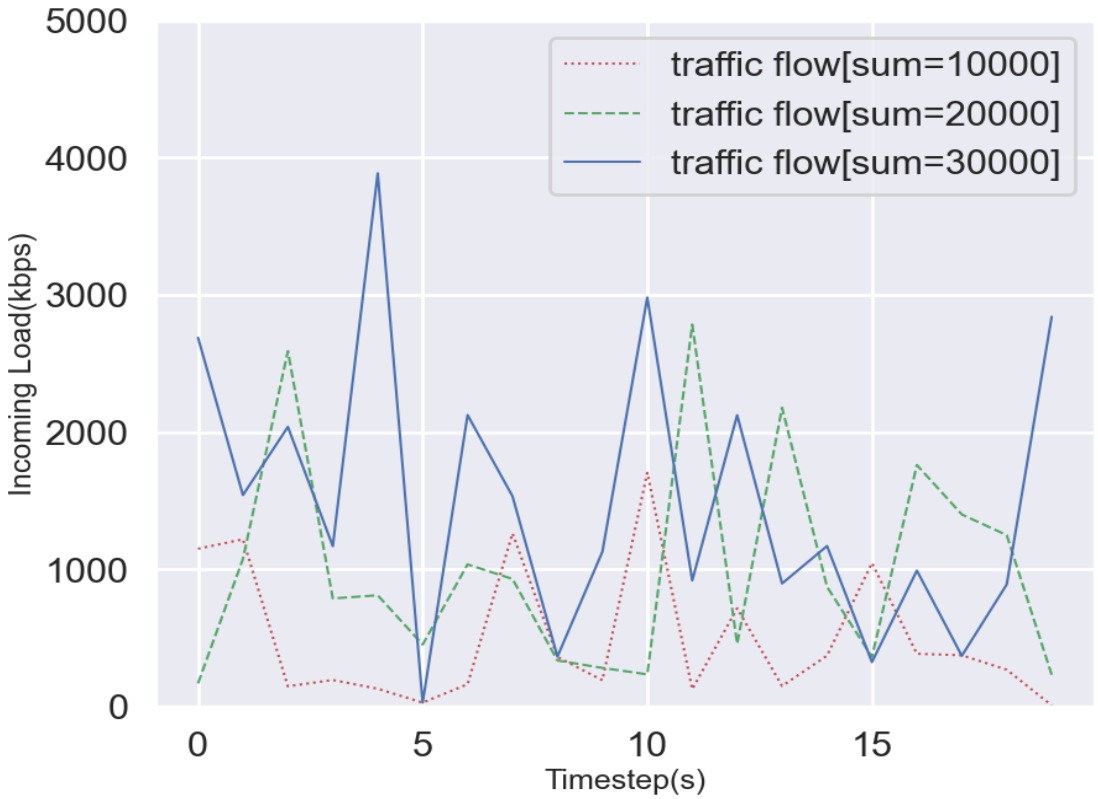

Figure 7 Random generation of traffic flow

The controller processing rate of the DALB, CAMD, ISMDA and the proposed MIBLB algorithm are shown in Figs. 8, 9, 10 and 11 respectively. The graph showed that more flow requests were successfully handled by the proposed MIBLB than the ISMDA and other algorithms. The previous study [6] proved that the ISMDA approach perform better when it is

compared to the DALB and CAMD approach. Hence, this study focuses on comparing the ISMDA with the proposed MIBLB algorithm.

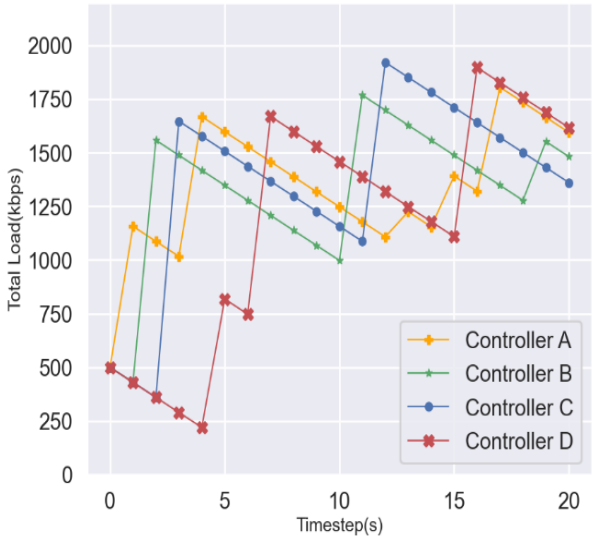

Figure 8 DALB controller processing rate

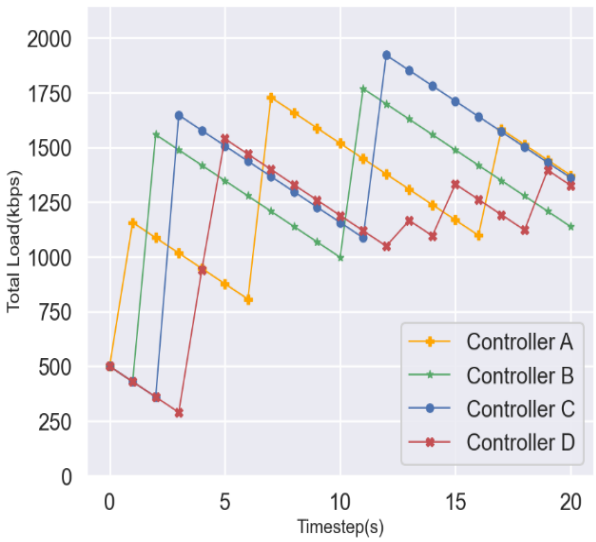

Figure 9 CAMD controller processing rate

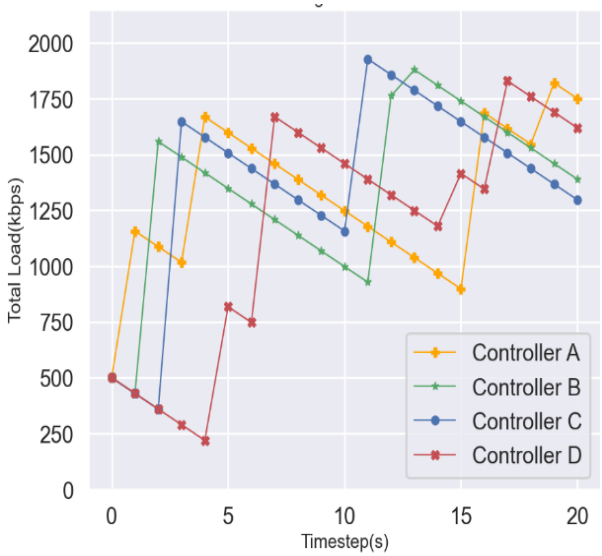

Figure 10 ISMDA controller processing rate

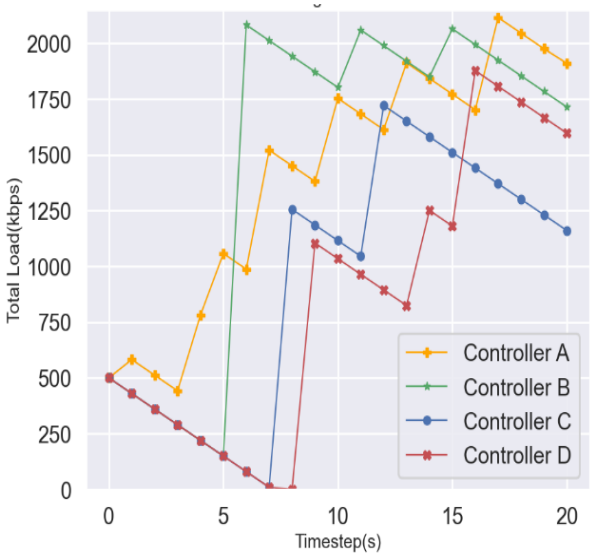

Figure 11 Proposed MIBLB controller processing rate

The average throughput comparisons of the four algorithms with different incoming load and numbers of controllers (two controllers, four controllers and eight controllers) are shown in Figs. [12-14]. It shows that the MIBLB approach successfully handled more incoming flow requests than ISMDA. The percent improvement for throughput, migrations, and response time is calculated as follows:

$$\text{Percent of improvement} = \frac{(ESMDA - ISMDA) * 100}{ISMDA} \qquad (10)$$

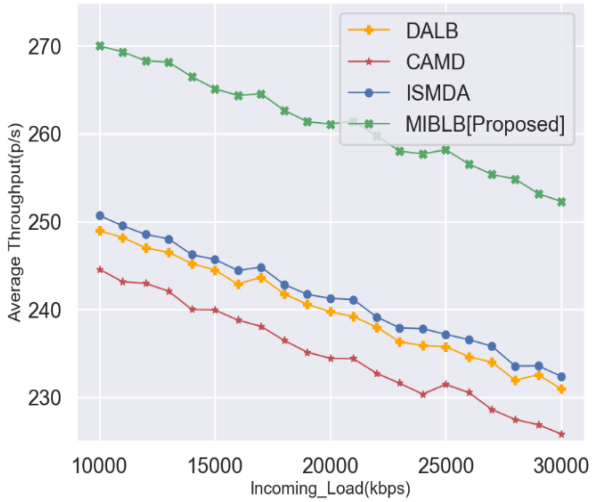

Figure 12 Average throughput of two controllers

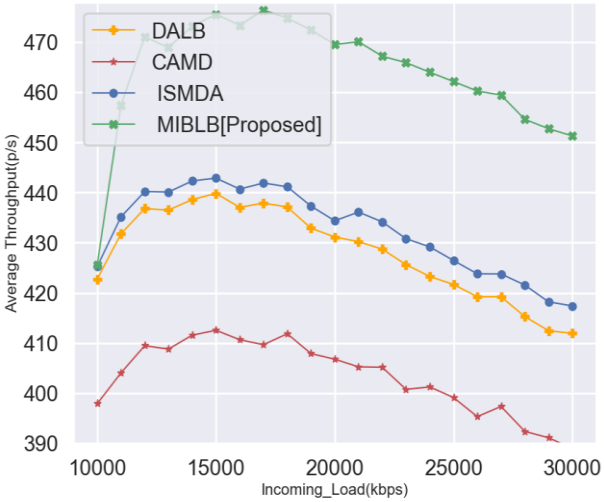

Figure 13 Average throughput of four controllers

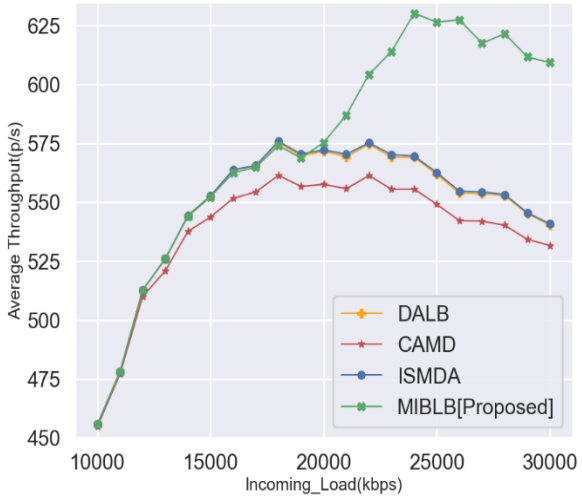

Figure 14 Average throughput of eight controllers

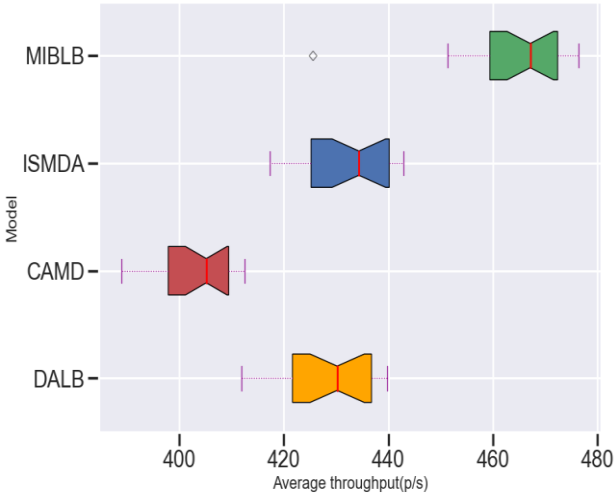

Fig. 15 Throughput of different algorithm by box plot

When compared to ISMDA, MIBLB has more processing capability at peak load and a low rate of packet loss. When the number of controllers is large and the incoming load is low, the controllers become idle and the throughput of the controller becomes nearly the same for both the ISMDA and MIBLB algorithms until the traffic becomes high. This implies that the efficiency of the MIBLB is more observable when the packet-in traffic is under the situation of high traffic flow. Figure 15 shows the results of the average throughput of various algorithms in minimum, the first 25% quarter(Q1), median, the third 75% quarter (Q3), and maximum values. The box plot clearly shows the distribution result of the throughput in case of minimum, maximum, median, and quarter values. It reveals that the median of the proposed algorithm is around 468p/s while the baseline algorithm is about 434p/s. The minimum and maximum throughput results of the MIBLB are 450p/s and 475p/s respectively while the ISMDA minimum and maximum throughput results are 415p/s and 443p/s respectively. This shows

that the proposed MIBLB algorithm is more efficient than the baseline ISMDA load balancing algorithm.

No of Controller	Incoming load	Average throughput(p/s)		Improvement in percent(%)
		ISMDA	MIBLB	
2	Low	245.68	265.69	8.14
	High	236.42	256.58	8.52
4	Low	439.13	466.33	6.19
	High	426.60	460.63	7.97
8	Low	538.45	537.73	0.13
	High	561.16	617.15	9.97
Average		407.90	434.01	6.82

Table 4 Comparison of MIBLB & ISMDA throughput

Table 4 shows that the proposed MIBLB increases the average controller throughput to about 434.01p/s, an increase of 6.82% over the ISMDA method. From the throughput comparison of MIBLB and ISMDA, it is demonstrated that MIBLB outperforms ISMDA algorithms.

4.3.2. Response time

The length of time it takes for a user to retrieve the results of a request is described as response time. For calculating response time, this study used Equation 11, which was created by Netforecast [35]. It was concerned with calculating the result of packet loss on response time. The response time equation is given as:

$$R = Rd + Rt \tag{11}$$

where

$$Rd = 2[D + Cc + Ct] + \left[D + \left(\frac{Cc + Cs}{2}\right)\right]\frac{T-2}{m} + Dln\left[\frac{T-2}{m} + 1\right] + KT\left(\frac{L}{1-L}\right) \tag{12}$$

and

$$Rt = \frac{MAX\left[8p\frac{1+OHD}{B} * Dpw\right]}{1-\sqrt{L}} \tag{13}$$

In Equation 11, R denotes the response time, *Rd* represents propagation delay time, and *Rt* represents the transmission delay time. In Equation 12, D represents round-trip delay, *Cc* represents current processing time, *Ct* represents server TCP processing, *Cs* represents server processing time, T represents application turns, *m* represents multiplexing factor, *Dln* represents packet-loss ratio and K represents TCP timeout. Additionally, in Equation 13, P represents payload length, *OHD* represents overhead ratio, *B* represents minimum path bandwidth and *W* represents the effective window size.

In this study the simulation of four algorithms are done to compare average response time of the controllers. The average response time of these four algorithms increase when an incoming load increases. Figs. [16-18] show that the proposed MIBLB algorithm improved the ISMDA in terms of response time. This is because at any stage of migration, MIBLB chooses most appropriate controller and give priority to mostly loaded controller so that the network congestion and packet rejection are avoided. As a result, the MIBLB algorithm ensured that resources were used efficiently by reducing the amount of rejected packets and controller response time.

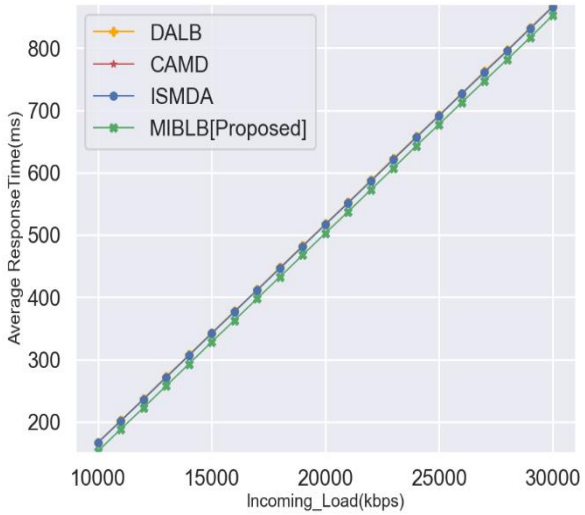

Figure 16 Average response time of two controllers

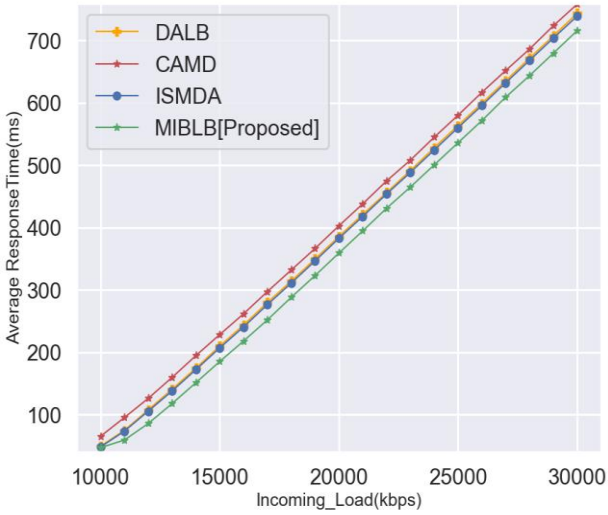

Figure 17 Average response time of four controllers

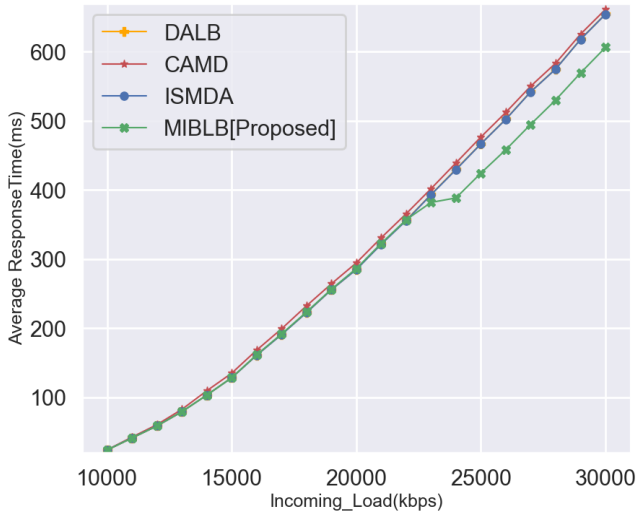

Fig. 18 Average response time of eight controllers

No of controller	Incoming load	Average response time		Improvement in percent
		ISMDA	MIBLB	
2	Low	341.26	327.66	-3.98
	High	708.63	694.79	-1.95
4	Low	208.75	195.54	-6.32
	High	578.42	561.52	-2.29
8	Low	141.03	141.01	-0.01
	High	485.61	453.25	-6.66
Average		**410.61**	**395.62**	-3.53

Table 5 Comparison of MIBLB & ISMDA response time

Table 5 shows that the average response time of the MIBLB algorithm is negative or reduced by 3.53% than the ISMDA algorithm. The negative results for response time indicate improvement because load balancing efficiency increases with a reduction in response time.

4.3.3. Number of switch migration

Number of switch migration indicates how many times each algorithm migrates a switch when a controller is overloaded. Figs [19-21] show that the average number of times each algorithm had to conduct the migration process. The simulation clearly demonstrates that an increase in incoming load would cause the average number of migration to increase for all four algorithms.

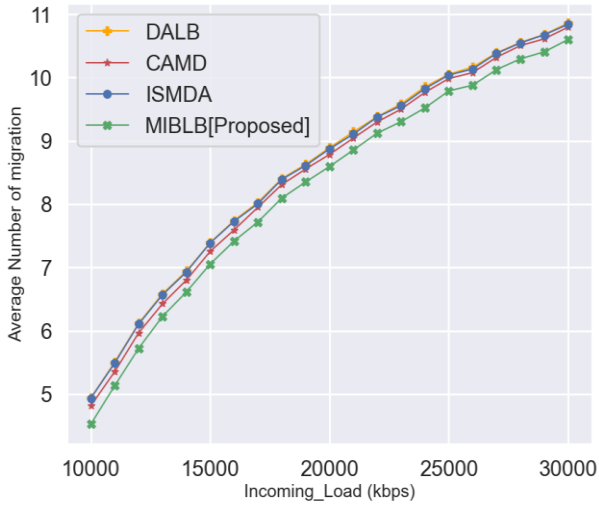

Figure 19 Number of migration of two controllers

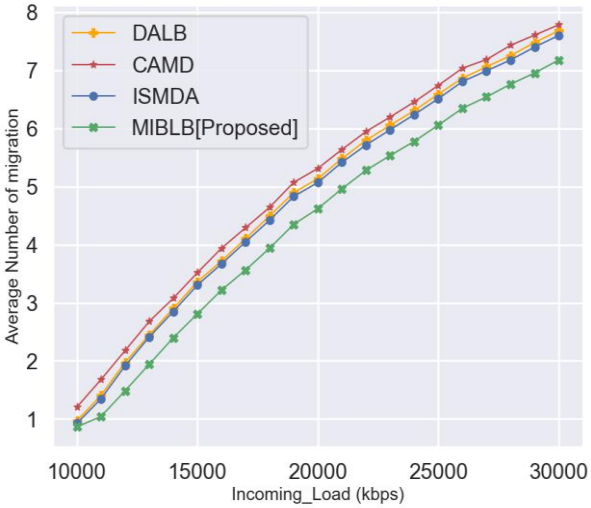

Figure 20 Number of migration of four controllers

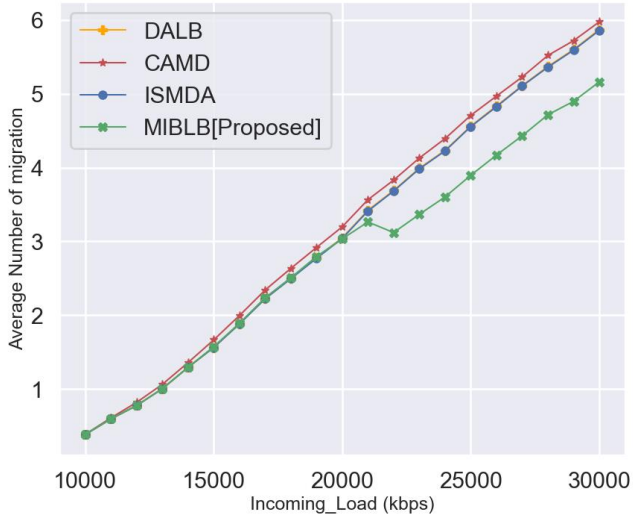

Fig. 21 Number of migration of eight controllers

No of con troller	Incoming load	Average number of migration		Improve ment in percent
		ISMDA	MIBLB	
2	Low	7.19	6.87	-4.45
	High	10.07	9.80	-2.68
4	Low	3.15	2.75	-12.69
	High	6.56	6.12	-6.70
8	Low	1.63	1.63	0
	High	4.64	4.08	-12.06
Average		5.54	5.20	-6.43

Tab. 6 Comparison of MIBLB and ISMDA No of migration

From Table 6, we can observe that the MIBLB algorithm performs better than ISMDA by reducing number of switch migration by 6.43% on average. The negative result shows that the number of switch migration is reduced, as expected from an efficient load balancing algorithm. This implies that, compared to ISMDA and other algorithms, the MIBLB will result in fewer switch migrations under controller load imbalance. The effective technique used and the selection of the appropriate controller at each stage of migration result in a decrease of the average number of switch migrations.

4.3.4. Discussion

This paper offers the MIBLB load balancing method for the SDN control plane based on the multi-level load interval to identify controller load status accurately. This approach solves the limitations of ISMDA by applying a dynamic threshold and migrating the switch from the overloaded controller to the appropriate under-loaded controller without causing a high migration cost or response time. The summary of the simulation's output obtained from the experiment has been calculated and shown in Table 7 for comparison. For evaluation purposes, the simulation iterates 1500 times to increase the reliability of the result. The same experiment with two, four and eight number of controllers is conducted by applying low and high incoming loads. At first, it can be observed that as the incoming load of controller increase, the response time and number of switch migrations will also increase. But the controller's throughput will increase until the incoming load reaches the maximum threshold, at which point it will start declining.

No of controllers	Throughput improvement (%)	Response time improvement (%)	Number of migration improvement (%)
Two	8.33	-2.96	-3.56
Four	7.08	-4.30	-9.69
Eight	5.05	-3.33	-6.03
Average	**6.82**	**-3.53**	**-6.43**

Table 7 Summary of simulation result

During load imbalance and target controller selection, the proposed MIBLB algorithm considers the status of the overloaded controller, whether the overloaded controller is highly overloaded or not. If so, it will give priority to highly overloaded controllers, and it will skip computing migration efficiency, variance, and switch selection efficiency to reduce communication overhead and avoid network bottlenecks. The results of the simulation have shown that MIBLB algorithm reduces number of migrations and controller response time of ISDMA by 6.43% and 3.53%, respectively. The controller throughput of ISDMA is also increased by 6.82%. The negative results in terms of switch migration and response time indicate improvement because load balancing performance improves with decreasing of switch migration and response time. The improvements of MIBLB are relatively high in terms of throughput, number of migrations and response time when the incoming load is high and the

number of controllers is relatively low compared to network traffic, i.e., when the controller resources are fully utilized.

4.3.5. Comparative analysis

A comparative analysis depicts the difference between the existing (ISMDA) algorithm and the proposed (MIBLB) techniques and reveals why the proposed algorithm is better than other existing algorithms. The proposed technique is the enhanced version of the ISMDA load balancing mechanism using multi-level load interval. The proposed MIBLB load balancing algorithm chooses the target endpoint based on the calculated accurate load status of controllers in the domain and it gives priority for highly overloaded controller to migrate the switch to lightly loaded controller which will minimize the packet loss. This technique minimizes the searching time to find out the receiver controllers in order to transfer the load. The MIBLB method is adopted to enhance the efficiency of the load balancing model.

Techniques	Existing technique(ISMDA)	Proposed technique(MIBLB)
Load interval categories	Divided into two intervals like: under-loaded and overloaded	Divided into four intervals like: idle, normal, overloaded, and highly overloaded.
Techniques	Each heavily loaded controller has to check all the nodes in the under-loaded pool	Each overloaded controller is checked into the pool either idle or normal controllers pool
Response time	High	Low
Throughput	Low	High
Number of migration	High	Low
Resource allocation capacity	low	High (due to the idea of load interval)
Cost	Low	High (due to computational resource usage)

Table 8 The difference between ISMDA and MIBLB techniques

5. Conclusions and Future Work

To solve the problem of load imbalance in multi-controller SDN, this paper proposes MIBLB algorithm that effectively addresses the challenge of load imbalance among multi-controller SDN controllers and avoid the incidents of controller overload. The MIBLB algorithm distinguishes the controller load status accurately by implementing a multi-level load interval that is based on the average controller load status. When the average controller load level rises, the proposed MIBLB algorithm dynamically adjusts the value of the default threshold. This will help to reduce packet loss and avoid network congestion in the network environment. Furthermore, the algorithm reduces the arising of communication overhead and migration costs by giving priority to mostly overloaded controllers in accordance with the estimated controller load balancing rate status. The experimental results show that MIBLB reduces the number of migrations and response time of ISDMA by 6.43% and 3.53% respectively and also increases throughput by 6.82%. This implies that compared to other algorithms, the MIBLB algorithm has high throughput, a low response time, and a low number of switch migrations. This is because the MIBLB algorithm is more efficient than other algorithms in terms of estimating accurate controller load status and giving priority to the highly overloaded controller, which reduces communication overhead and response time. In the future, this algorithm could be implemented on actual SDN controllers and large scale network environments.

6. References

[1] T. Hu, Z. Guo, T. Baker and J. Lan, "Multi-controller Based Software-Defined Networking: A Survey," *IEEE,* p. 21, 2017.

[2] O. Blial, M. B. Mamoun and R. Benaini, "An Overview on SDN Architectures with Multiple Controllers," *Hindawi,* p. 9, 2016.

[3] H. Amir, W. Michael and A. Kourosh, "An Overview of Multi-Controller Architecture in Software-Defined Networking," *IEEE,* p. 7, 2019.

[4] Y. Fu, Z. Yuting, C. Zijian, D. Zhiqiang, G. Yan and J. Du, "Multi-Controller Load Balancing Algorithm for Test Network," *MDPI,* p. 18, 2021.

[5] Y. Shike, H. Shi and H. Zhang, "Dynamic Load Balancing of Multiple Controller based on Intelligent Collaboration in SDN," *CVIDL,* p. 6, 2020.

[6] O. ADEKOYA, A. ANEIBA and M. PATWARY, "An Improved Switch Migration Decision Algorithm for SDN Load Balancing," *IEEE,* p. 12, 2020.

[7] H. M. Noman and D. N. Jasim, "A Proposed Linear Multi-Controller Architecture to Improve the Performance of Software Defined Networks," *ECS,* p. 11, 2021.

[8] M. R. BELGAUM, S. MUSA, M. M. ALAM and M. M. SU'UD, "A Systematic Review of Load Balancing Techniques in Software-Defined Networking," *IEEE,* p. 15, 2020.

[9] M. Hamdan, E. Hassan, A. Abdelaziz, A. Elhigazi, B. Mohammed, S. Khan, A. V. Vasilakos and M. Marsono, "A comprehensive survey of load balancing techniques in software-defined network," *Elsevier,* p. 42, 2020.

[10] K. Sajad, Sánchez, A. Gallego and A. S. Tosun, "Hybrid SDN evolution: A comprehensive survey of the state-of-the-art," *scienceDirect,* p. 40, 2021.

[11] M. Jammal, T. Singh, A. Shami, A. Rasool and Y. Li, "Software defined networking: State of the art and research," *scienceDirect,* p. 15, 2014.

[12] A. Montazerolghaem, "Software-defined load-balanced data center: design, implementation," *Springer,* p. 20, 2020.

[13] A. Shirmarz and A. Ghaffari, "Performance issues and solutions in SDN-based data center: a survey," *springer,* p. 49, 2020.

[14] Chen, M.-Y. Luo and Jun-Yi, "Software Defined Networking across Distributed Datacenters over Cloud," *IEEE,* p. 7, 2015.

[15] L. GUOYAN, W. XINQIANG and Z. ZHIGANG, "SDN-Based Load Balancing Scheme for Multi-Controller Deployment," *IEEE,* p. 11, 2019.

[16] H. Tao, L. Julong, Z. Jianhui and Z. Wei, "EASM: Efficiency-aware switch migration for balancing controller loads in software-defined networking," *Springer,* p. 13, 2018.

[17] L. Zhuo, Z. Xu, G. Junruo and Q. Yifang, "SDN Controller Load Balancing Based on Reinforcement Learning," *IEEE,* p. 7, 2021.

[18] M. Priyadarsini, C. M. Joy, P. Bera, S. Kumar, J. A. H. M and R. M. Ashiqur, "An adaptive load balancing scheme for software-defined network controllers," *ScienceDirect,* p. 11, 2019.

[19] S. K. Sagar, M. Senior, T. Mayank, U. Muhammad, S. Bibhudatta, Z. Wen, S. Biswa and R. Rajiv, "ESMLB: An Efficient Switch Migration based Load Balancing for Multi-Controller SDN in IoT," *IEEE,* p. 9, 2020.

[20] Y. Zhou, M. Zhu, L. Xiao, L. Ruan, W. Duan, D. Li and R. Liu, "A Load Balancing Strategy for SDN Controller based on Distributed Decision," *IEEE,* p. 6, 2014.

[21] S. S. Kshira and B. Sahoo, "CAMD: a switch migration based load balancing framework for software defined networks," *IET,* vol. 8, p. 8, 2019.

[22] Z. Yuan, C. Lin, W. Wei and Z. Yuxiang, "A Survey on Software Defined Networking with Multiple Controllers," *ScienceDirect,* p. 59, 2018.

[23] A. Rahmatollah, M. Hossein and M. Seyedakbar, "Load Balancing In Multi-Controller Software-Defined Networks," *ResearchGate,* p. 11, 2020.

[24] F. Chahlaoui and H. Dahmouni, "A Taxonomy of Load Balancing Mechanisms in Centralized and Distributed SDN Architectures," *Springer ,* p. 16, 2020.

[25] Z. SHAOJUN, L. JULONG, S. PENGHAO and J. YIMING, "Online Load Balancing for Distributed Control Plane in Software-defined Data Center Network," *IEEE,* p. 9, 2018.

[26] W. Fouad Aly and A.-a. Abeer, "Enhanced Controller Fault Tolerant (ECFT) Model for Software Defined Networking," *ResearchGate,* p. 7, 2018.

[27] Y. Xing, C. Guozhen and L. Xingguo, "Maximizing SDN Control Resource Utilization via Switch Migration," *Elsevier,* p. 14, 2017.

[28] L. Wenjing, L. Fangmin, L. Xinhua and Q. Yiwen, "A Dynamic Load Balancing Mechanism for Distributed Controllers in Software-Defined Networking," *IEEE,* p. 4, 2018.

[29] C. WANG, B. HU, S. CHEN, D. LI and B. LIU, "A Switch Migration-Based Decision-Making Scheme for Balancing Load in SDN," *IEEE,* p. 8, 2021.

[30] Z. Hong, F. Jinpeng, C. Jie, X. Yan and L. Liu, "Assessing Profit of Prediction for SDN controllers load balancing," *Elsevier,* p. 10, 2021.

[31] K. Raj, S. Abhishek, T. Mayank and S. K. Sagar, "Improving Quality of Services During Device Migration in Software Defined Network," *Springer,* p. 8, 2018.

[32] N. Amit, S. A. Kumar and A. L. Kumar, "Laman: A Supervisor Controller based Scalable Framework for Software Defined Networks," *Elsevier,* p. 2019, 17.

[33] F. AL-TAM and N. CORREIA, "On Load Balancing via Switch Migration in Software-Defined Networking," *IEEE,* p. 9, 2019.

[34] C. Min, "Load Balancing of SDN Controller for Migration Optimization Based on Metaheuristics," *Hindawi,* p. 10, 2022.

[35] M. Miyagi, K. Ohkubo, M. Kataoka and S. Yoshizawa, "Performance Prediction Method for Web-Access Response Time Distribution Using Formula," *IEEE,* vol. I, pp. 905 - 906, 2004.

[36] T. Hu, Z. Guo, T. Baker and J. Lan, "Multi-controller Based Software-Defined Networking: A Survey," *IEEE,* p. 21, 2017.

[37] O. Blial, M. B. Mamoun and R. Benaini, "An Overview on SDN Architectures with Multiple Controllers," *Hindawi,* p. 9, 2016.

[38] A. Hossein, M. Watts and K. Ahmadi, "An Overview of Multi-Controller Architecture in Software-Defined Networking," *IEEE,* p. 7, 2019.

[39] Z. Yuanhao, Z. Mingfa, X. Limin, R. Li, D. Wenbo, L. Deguo and L. Rui, "A Load Balancing Strategy for SDN Controller based on Distributed Decision," *IEEE,* p. 6, 2014.