Tribhuvan University

INSTITUTE OF ENGINEERING

PULCHOWK, LALITPUR

Dissertation No: 078PhCT501

Multi-Controller Placement Optimization with Load Balancing in Software-Defined Networking

Binod Sapkota

A DISSERTATION SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

February, 2026

Dedicated to my Family and Parents . . .

Bhimsen Prasad Sapkota (Father)
Jamuna Kumari Sapkota (Mother)
Kalpana Sapkota (Wife)
Bikalpa Sapkota (Son)
Aayan Sapkota (Son)

Copyright©

Declaration of Authorship

The dissertation entitled "**Multi-Controller Placement Optimization with Load Balancing in Software-Defined Networking**", which is being submitted to the Department of Electronics and Computer Engineering, Pulchowk Campus, IOE, TU, Nepal, for the award of the degree of Doctor of Philosophy (PhD) in Computer Engineering is a research work carried out by me under the supervision of Prof. Dr. Shashidhar R. Joshi, Department of Electronics and Computer Engineering, IOE, Pulchowk Campus, TU, Nepal, and Asst. Prof. Dr. Babu R. Dawadi, Department of Electronics and Computer Engineering, IOE, Pulchowk Campus, TU, Nepal, between February 2022 and February 2026. I declare that this is my work and has not been previously submitted by me at any university for any academic award.

Binod Sapkota

Signed:

_____

February 17, 2026

Recommendation

The undersigned certify that they have read and recommended for acceptance, a dissertation entitled **"Multi-Controller Placement Optimization with Load Balancing in Software-Defined Networking"**, submitted by **Binod Sapkota** in partial fulfillment of the requirement for the award of the degree of **Doctor of Philosophy in Computer Engineering**.

Prof. Shashidhar Ram Joshi, PhD
Supervisor,
Department of Electronics and Computer Engineering, Pulchowk Campus, IOE,
Tribhuvan University, Nepal

Asst. Prof. Babu R. Dawadi, PhD
Supervisor,
Department of Electronics and Computer Engineering, Pulchowk Campus, IOE,
Tribhuvan University, Nepal

Er. Kamal Lamichhane, PhD
Manager, Nepal Telecom,
External Examiner, DRC

April 09, 2025

Departmental Acceptance

The doctoral dissertation entitled **"Multi-Controller Placement Optimization with Load Balancing in Software-Defined Networking"**, submitted by **Binod Sapkota** in partial fulfilment of the requirement for the award of the degree of **Doctor of Philosophy in Computer Engineering** has been accepted as a bonafied record of work carried out by him in the department.

Prof. Ram Krishna Maharjan, PhD
Head of Department
Department of Electronics and Computer Engineering, Pulchowk Campus,
Institute of Engineering,
Tribhuvan University, Nepal
April 09, 2025

Tribhuvan University
INSTITUTE OF ENGINEERING

The undersigned certify that they have evaluated the dissertation entitled **"Multi-Controller Placement Optimization with Load Balancing in Software-Defined Networking"** submitted by **Binod Sapkota** and have external oral presentation for the partial fulfillment of the requirement for the degree of Doctor of Philosophy in Computer Engineering and recommended to IOE for the acceptance of this dissertation.

Prof. Devesh Kumar Srivastava, PhD
Department of Information Technology,
Manipal University, Jaipur, India
External Examiner

Prof. Sudan Jha, PhD
Department of Computer Science and Engineering,
Kathmandu University, Nepal
Internal Examiner

Er. Bimal Acharya, PhD
Former Manager,
Nepal Telecom, Nepal
Internal Examiner

Tribhuvan University
INSTITUTE OF ENGINEERING

The dissertation **"Multi-Controller Placement Optimization with Load Balancing in Software-Defined Networking"** submitted by **Binod Sapkota** for partial fulfillment of the requirement for the degree of Doctor of Philosophy in Computer Engineering has been accepted by the IOE Research Committee (IERC) upon the recommendation of the supervisor and Departmental Research Committee (DRC) with the approval by the following examiners.

External Examiner:

Prof. Devesh Kumar Srivastava, PhD
Department of Information Technology,
Manipal University, Jaipur, India

Internal Examiners:

Prof. Sudan Jha, PhD
Department of Computer Science and Engineering,
Kathmandu University, Nepal

Er. Bimal Acharya, PhD
Former Manager,
Nepal Telecom, Nepal

Prof. Sushil B. Bajracharya, PhD
IERC Chairperson and Dean,
Institute of Engineering

February 17, 2026

Abstract

Software-Defined Networking (SDN) is the novel networking paradigm where decoupling of the control plane from the data plane has its inherent advantages. With the successful implementation of Software-Defined Networking in data center networking, the way forward for its deployment in the ISP/Telco network is becoming prominent. The research on SDN deployment and implementation for data centres and ISP/Telco is still continuing. The Controller Placement Problem involves placing the optimal number of controllers at the appropriate locations to achieve better performance (such as latency minimization, load balancing, energy efficiency, and enhanced reliability). To achieve scalability, the deployment of multiple controllers on a large scale in Software-Defined Networking is one of the key challenges.

In this regard, the current research status of the Multi-Controller Placement Problem over Software-Defined Networking with their specific challenges and a comprehensive review of the major performance metrics, i.e., latency and controller load balancing techniques, are properly presented. A novel population-based metaheuristic algorithm, viz. The Naked Mole Rat algorithm, has been tested to optimize the location for controller placement based on switch-to-controller, controller-to-controller latency while maintaining load balancing among the controllers. The ideas and mechanisms are illustrated using two publicly available standard topologies, viz. Ernet and Savvis. The controller localization approach implemented with the Naked Mole-Rat (NMR) algorithm has better results compared to the Bat algorithm.

These days, researchers world-wide are more interested towards traffic engineering in Software-Defined Networking after its conceptual design and implementation. As internet usage continues to rise, the ability to identify and manage network traffic based on its priority has become a critical requirement. Even with multiple controllers deployed to handle the growing volume of data from intelligent devices, dynamic traffic patterns can still lead to unbalanced load distribution across controllers. Machine Learning has shown significant promise in enabling accurate traffic classification and optimal controller selection. Therefore, different classification models are

evaluated to identify the most effective approach for real-time traffic classification. Among the models tested, the Classification and Regression Tree (CART) classifier achieved the best performance on the generated dataset, while logistic regression also demonstrated competitive results.

Furthermore, based on the evaluation of various algorithmic outputs for the training and validation dataset, and when execution time is taken into account, the classification and regression tree is found to be the best algorithm. During the process of evaluating the effects of load balancing in a multi-controller Software-Defined Networking system, several load case scenarios are examined, and bit rate, packet rate, and jitter are measured. Here, the use of traffic classification-based load balancing improves the bit rate and the packet rate of traffic flow on a network and thus considerably enhances throughput. Finally, the reduction in jitter while increasing the controllers confirms the improvement in Quality of Service in a balanced multi-controller Software-Defined Networking environment.

Moreover, the increasing complexity and scale of modern software-defined networking demand advanced solutions to address security challenges, particularly distributed denial-of-service attacks in multi-controller environments. Traditional single controller implementations are struggling to effectively counter sophisticated cyber threats, which require a faster and scalable solution. When a distributed denial of service attack occurs, the network balance is disturbed. DDoS attacks pose a significant threat to multi-controller SDN environments, disrupting network services, and overwhelming resources. Traditional detection and mitigation techniques often do not handle the complexity and dynamic nature of these attacks. Integrating ML into SDN can improve near-real-time attack detection and mitigation, improving security.

Therefore, the design, implementation, and assessment of optimal placement of multi-controllers using K-means++ and OPTICS in real topologies are focused on. An Intrusion Detection System using the XGBoost classification algorithm within the software-defined networking environment is highlighted to detect and mitigate distributed denial-of-service attacks. Additionally, the Intrusion Detection System decouples from the controllers, preserves controller resources, and allows for efficient near-real-time attack detection and mitigation. The proposed solution overcomes previous research by autonomously identifying anomalous behaviors in the network by successfully combining the Controller Placement Problem and Distributed Denial-of-Service attacks security.

Additionally, the best strategy for placing multiple (newly added) SDN controllers using a nature-inspired algorithm, i.e., naked-mole-rat, to solve multicontroller placement issues, an intelligent traffic-driven controller load balancing strategy for multiple controllers applying machine learning and the SDN security issues of attack detection and mitigation in the data plane of the optimized multicontroller topologies are appropriately highlighted in this research. The experiment results perform better in each scenario. During the literature review, some key ongoing research areas and the future research direction regarding the various software-defined networking-based controller placements in advanced next-generation networking technologies have also been discussed.

Finally, to enhance the proposed system's performance further, there is still room for improvement. Researchers might be able to improve the network by testing the proposed approach on larger and more complex topologies, including real-world scenarios; adding more traffic types and applications for more complex classification; using intelligent traffic classification and prediction in SDN; testing the model for present NetworkX topologies; and integrating it with edge computing, cloud services, and network slicing.

At present, the majority of researchers are working on the controller placement problem for data centres. However, they do not offer a particularly efficient approach for the large-sized ISP/Telco networks with multi-controller dynamic traffic loads and faults. The efficient and optimal choice of numbers and placement of SDN controllers is a challenge that might be solved intelligently through further research in this field. Therefore, further research is needed to improve the network's performance, availability, and reliability while making it more balanced and safer.

Keywords: Software-Defined Networking, Controller Placement Problem, Naked Mole-Rat, Traffic Classification, Load balancing, Distributed Denial-of-Service attacks, Machine Learning

Acknowledgement

With immense pleasure, I would like to express my sincere gratitude to my advisors, **Prof. Dr. Shashidhar R. Joshi**, former Dean of Institute of Engineering, Tribhuvan University, Nepal, and **Asst. Prof. Dr. Babu R. Dawadi**, Asst. Dean (Administration) of Institute of Engineering, Tribhuvan University, Nepal, for their continuous and enormous support during my Ph.D. study. I felt very lucky to have such patient, motivated, and knowledgeable professors as my supervisors, whose fruitful guidance, in fact, properly landed me with such a great recognition in my life. Undoubtedly, I had always sought much more advice and assistance from them during the research.

In addition to my full-time advisors, I am deeply indebted to my visiting advisor, **Asst. Prof. Dr. Sidharth Sharma**, Department of Computer Science and Engineering, Indian Institute of Technology Indore (IITI), India, for his unconditional support and facilitation during my short-term collaborative research visit (DEC 26, 2024 - Feb 02, 2025). During my research, I have had wonderful working experiences with such an excellent professor on research projects and possible publications, and I hope to continue our collaboration in the future.

My sincere thanks go to **Prof. Dr. Ram Krishna Maharjan**, Head of Department, **Assoc. Prof. Dr. Jyoti Tandukar**, chairman of the DRC, other DRC members Assoc. Prof. Dr. Arun K. Timalsina, Asst. Prof. Dr. Aman Shakya, Asst. Prof. Dr. Basanta Joshi, Assoc. Prof. Dr. Sanjaya Uprety (Campus chief), and Assoc. Prof. Dr. Indra Acharya (former campus chief) for their support and evaluation with fruitful comments from the proposal to date.

I am also grateful to Prof. Dr. Subarna Shakya, Assoc. Prof. Dr. Sanjeeb Prasad Panday, Assoc. Prof. Dr. Nanda Bikram Adhikari, Assoc. Prof. Dr. Dibakar Raj Pant, and many other faculties and colleagues for their constructive comments, direct or indirect support, and encouragement for my research. And, with no doubt, their insightful suggestions and criticism will be desired and welcomed until the research is complete.

It is my pleasure to express gratitude to Asst. Prof. Dr. Khem Gyanwali, Campus Chief of Thapathali Campus; Asst. Prof. Janardan Bhatta, former Campus Chief of Thapathali Campus; and Asst. Prof. Kiran Chandra Dahal, former Head of Department of Electronics and Computer Engineering at Thapathali Campus, whose motivation and support are highly appreciated. Nevertheless, I am also appreciative of Dr. Samundra Thapa, who supported me by providing the latest journal paper, whenever required.

I am thankful to the University Grant Commission (UGC), Nepal, for granting me permission to publish research papers under a research grant (Grant ID: CRG-078/79-Engg-01). That grant undoubtedly improved the research and inspired me to carry out more in the future.

Of course, I am grateful to my family and parents for their moral support and inspiring motivation. I don't consider myself without them. Special gratitude to my sister Rupa Sapkota and brother-in-law, Dr. Ramesh Devkota, who gifted me a powerful laptop to carry out my PhD study with the positive hope of success. Last but not least, I would like to acknowledge my wife, Kalpana Sapkota, for her emotional support and encouragement that have brought me where I am today.

Binod Sapkota
078PhCT501

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ABFO	Adaptive Bacterial Foraging Optimization
API	Application Programming Interface
BFO	Bacterial Foraging Optimization
BFR	Breadth First search Routing
CC	Controller to Controller
CNPA	Clustering-based Network Partition Algorithm
CP	Conference Paper
CPL	Control Plane Latency
CPP	Controller Placement Problem
DALB	Distributed Algorithm and Load Balancing
DDoS	Distributed Denial of Service
DE	Differential Evolution
DITG	Distributed Internet Traffic Generator
DL	Deep Learning
ESCALB	Effective Slave Controller Allocation-based Load Balancing
FFA	Fire Fly Algorithm
FPA	Flower Pollination Algorithm
GSA	Gravitational Search Algorithm
GWO	Grey Wolf Optimization
JP	Journal Paper
LDOP	Load Distribution on controller Performance
LPA	Label Propagation Algorithm
LP/QP	Linear and Quadratic programming
LRCP	Latency and Reliability-aware Controller Placement
MCPP	Multi Controller Placement Problem
MCPO	Multi Controller Placement optimization
ML	Machine Learning
NMR	Naked-Mole Rat Algorithm
ODL	Open Daylight
OF	Open Flow
ONOS	Open Network Operationg System
OPTICS	Ordering Points to Identify the Clustering Structure
PAM	Partition Around Medoids
PECP	Path Computation Element Protocol
PSO	Particle Swarm Optimization
RCP	Reliability-aware Controller Placement
REST API	REpresentational State Transfer API
RL	Renforcement Learning

RP	Research Paper
SC	switch to Controller
SCPLBS	Smart Cooperative Platform for Load Balancing and Security
SDN	Software-Defined Networking
SODIP6	Software defined IPv6
SL	Supervised Learning
TLBO	Teaching-learning-based optimization
UL	Unsupervised Learning
VANETs	Vehicular ad hoc networks
VBO	Varna-based Optimization
WAN	Wide Area Network
WOA	Whale Optimization Algorithm
WCSS	Within-Cluster Sum of Squared

Contents

This page is left intentionally blank...

Chapter 1

Introduction

The research is introduced in this chapter with the background and statement of the problem. After designing research questions, research objectives are defined. List of publications throughout the study period are presented, and the overall structure of the thesis is outlined in this chapter.

1.1 Background

Software-Defined Networking (SDN) is a new paradigm that enables the dynamic nature of networks and smart services while offering simple network management allowing consumers to have a higher quality of experience and lowering expenses during its implementation, operation and maintenance [1], [2]. To handle a whole network, it isolates the control plane from the data plane, enabling network scalability and programmability. The control plane and data plane are interconnected in traditional network architecture [3], [4].

One of the most significant advantages in SDN is that network intelligence is centralized in software-based controllers [5]. SDN transforms the legacy switches into pure forwarding elements whose flow tables are populated by the controller.

Also, SDN is a modern networking approach that is flexible, manageable, cost-effective, and adaptable. The objective of SDN is to simplify network management, promote innovation, improve network resource utilization, and optimize network performance. It is well suited for high-bandwidth, dynamic applications. In large networks with hundreds of switches and routers, network management is a highly difficult and time-consuming procedure. Thus, SDN, a novel approach, seeks to improve the present unfavorable scenario for building, developing, and maintaining

networks. The core concept of SDN is to logically centralize network control in an SDN controller, which controls and monitors network behavior [6].

The traditional methods of configuring, optimizing, and troubleshooting computer networks may not be sufficient or efficient due to the large size, heterogeneity, and complexity of current and future networks [7]. SDN offers security, energy efficiency, and network virtualization for enhanced network performance as a result of the rising prevalence of smart programmable devices in networks [8]. Even though SDN provides many benefits, such as scalability, agility, and ease of administration, security is one of the most important aspects to take into account while using it [9].

The benefits of SDN [8] and the explanations for its rapid acceptance are depicted in Figure 1.1.

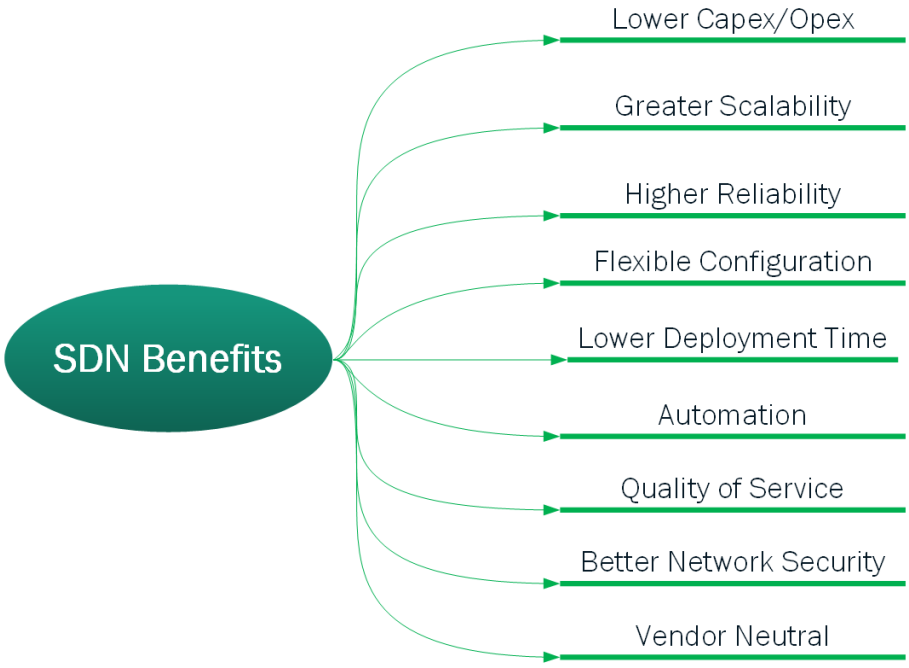


Figure 1.1: SDN benefits

Since the SDN architecture is centralized, programmable, and able to collect real-time data from the controller, it is conceivable to use some "intelligence" (machine learning techniques) for efficient routing and QoS provisioning [10]. By utilizing SDN's programmability, ML may be employed to offer real-time QoS solutions.

In SDN, the control and forwarding functions of a network are separated, enabling the control of the network to be programmable and the infrastructure for applications and network services to be abstracted. This design makes SDN a suitable choice for current and future networking needs.

SDN architecture consists of majorly three layers, (a) the application layer, (b) the control layer, and (c) the infrastructure layer. Each of these layers performs certain functions and communicates with the others via interfaces [11].

1.1.1 Multi-controller placement optimization

The Controller Placement Problem (CPP) involves determining the optimum number of controllers and their placement [12]. The CPP is an urgent problem that needs to be solved and asks how many controllers are needed and where they should be positioned [13]. The type, number, and location of these controllers directly affect the cost and average latency of control packets [14].

CPP has different components. The number of controllers to be placed in a network, their locations, and the function of assigning controllers to switches are all important to the design of an effective control plane [15].

The location of controllers in a distributed SDN controller architecture has a significant impact on network performance in terms of latency, reiability, and other factors, which is known as the CPP [16]. The number of controllers to be deployed and their placement are two facets of the CPP [17].

After conducting a thorough review of the literature, Sood et al. [18] conclude that there is no rigid placement rule in SDN that can be applied to every network. Dynamically adding and removing controllers is inevitable.

Ali et al. [19] indicate that the CPP is the process of determining the best places to put controllers and the best switches to go with those controllers in order to accomplish an objective. Researchers and network engineers have focused a lot of attention on one of the most challenging difficulties in SDN, the positioning of the controller.

Small and medium-sized networks don't have a CPP problem because they just need one controller to control them. However, in the context of increasing network size, there would be significant aspects to consider regarding the placement of multiple controllers [17]. Therefore, the CPP in a WAN mainly deals to divide the network into multiple sub networks and the minimum number of controllers and their deployments in each domain.

The CPP study was initially conducted by Heller et al. [20]. They defined the issue as a facility placement problem and demonstrated that it was NP-hard. Since then,

many initiatives have been made to position the controllers in the best possible way. By calculating the optimal location for the deployed controllers and the minimal number of controllers to be placed, the authors provided a principal objective to solve the CPP by reducing the propagation delay (latency).

To solve the NP-hard problem, efficient meta-heuristic algorithms such as PSO, FFA, VBO, k-means, GWO, BFO and so on for latency measurements criteria have been studied and designed for different networks [21].

Salgotra et al. [22] formulate the swarm intelligent nature-inspired algorithm called Naked Mol-Rat (NMR) algorithm and its performance is evaluated by a comparative study with other algorithms. Their experimental results and statistical analysis prove that NMR algorithm is very competitive as compared to other advanced algorithms like PSO, GWO, WOA, DE, GSA, FEP, Bat algorithm, FPA, and FFA.

1.1.2 Traffic-driven multi-controller load balancing

Load balancing in SDN is a crucial component of network management which is a technique that spreads the traffic over several resources in order to prevent overload on any of the resources [23]. It entails an effective distribution of network resources to enhance efficiency and prevent bottlenecks. Recently, SDN load-balancing methods have become more and more popular because of their affordability, scalability, and flexibility. Appropriate load balancing also aids in increasing throughput, decreasing response times, minimizing utilization of resources, preventing resource overload, and maximizing scalability. Although load-balancing research for SDN has advanced, to achieve better outcomes and greater adoption, there are still a number of issues and challenges, for example, multi-controller load-balancing, that need to be resolved [24].

Controller overload, scalability, security, traffic engineering, service-level agreements, dynamic load balancing for multiple controllers are common issues and challenges in SDN load balancing. Designing effective and efficient traffic algorithms can manage intricate network topologies and a variety of traffic patterns.

Gasmelseed et al. [25] propose a new load-balancing technique for SDN based on the traffic pattern that separates the traffic into TCP and UDP. The distributed controller design used in this study addresses the management, scalability, and availability limitations of the centralized controller.

Traditional methods like port number identification and payload inspection are ineffective because today's traffic is dynamic and encrypted [26]. The access control list for IP addresses and port numbers associated with various applications is typically updated manually in the traditional port-based methodology of classification, which can be a slow and challenging method of improving network QoS even though it may have high accuracy.

In the real world, not all network traffics are equally crucial. For instance, for some organizations, voice and message traffic may be more significant than game and video traffic, yet video streaming and gaming data may be significant for other organizations. This situation leads to poor performance of the network and load imbalance.

Even for experienced professionals in the field employing cutting-edge techniques, processing and analyzing large dataset generated from vast Internet traffic volumes is challenging. Identification of the type of traffic supports managing the resources and allocating them to the critical application, thus ensuring an improvement in QoS and stability of the system. Due to the high volume of traffic moving across the internet, internet service providers (ISPs) and networking service providers (NSPs) are looking for a number of approaches to properly predicting the type of application flow that is now running through it [27].

Thus, developing a system that dynamically classifies network traffic and reroutes traffic based on the criticality of the data received is the main motivation behind the research. Modern machine learning (ML) advancements have proved effectiveness in prediction applications, particularly in the field of networking.

1.2 Statement of the Problem

As the number of controllers and their location of deployment has significantly impacted the performance of the network in multi-controller network architecture, the CPP has attracted the research community. Thus, in order to achieve several defined goals, such as latency minimization, load balancing, energy efficiency, and enhanced reliability, which is crucial to SDN's performance in large-scale networks, CPP seeks to determine the best position for the SDN controllers [28].

To achieve better performance, identifying the correct number of controllers and their best locations within the network is the primary challenge in SDN. Over the

years, several CPP techniques have been proposed, developed, and used to find the best solution; each has distinct goals, advantages, and drawbacks.

With the world's technological advancement and rapid shift to ICT, there has been an urge for computer networks with high bandwidth, accessibility, robustness, reliability, and dynamic management. The size, variety, and complexity of networks make traditional methods ineffective and possibly insufficient for setting up, improving performance, and resolving problems with computer networks. With a boom in the use of the internet, identifying and managing traffic loads based on the importance of different traffic types is becoming an increasingly important function of networks.

Researchers and practitioners have proposed several SDN load-balancing approaches. This ensures network management, performance, scalability, and high availability. By offering a better quality of service, the use of SDN load balancing makes networks more flexible and manages them better, which in turn makes application services for end users more efficient and adaptive [29].

A balanced network is necessary because dynamic network traffic might cause an unequal distribution of loads across controllers, though multiple controllers are used to handle the heavy traffic generated by various intelligent devices. When a distributed denial of service (DDoS) attack occurs, this balance is disturbed. DDoS attacks pose a significant threat to multi-controller SDN environments, disrupting network services, and overwhelming resources. Traditional detection and mitigation techniques often do not handle the complexity and dynamic nature of these attacks [30]. Integrating ML into SDN can improve near-real-time attack detection and mitigation, improving security.

Existing research mostly focuses on data centers and pays less attention to SDN-enabled ISP/Telco networks, i.e., the CPP issues related to the ISP/Telco networks have not yet been completely reviewed and properly explained in any other research. Hence, in the published review paper [24], CPP, MCPP (scalability of CPP), latency, and load balancing (among various performance metrics) are primarily focused, and finally presented the implementation feasibility of SDN in ISP/Telco networks.

Moreover, controller placement optimization and traffic-driven load balancing in the SDN context are major areas of this research study. Additionally, SDN security in a multicontroller-based SDN environment is discussed.

1.3 Research Questions

Three research questions (RQ) were designed to address the main problem outlined in this research. These are:

RQ1: What are the effective and efficient multi-controller placement optimization techniques in an SDN environment?

RQ2: What is/are the intelligent approach(s) for load balancing in a multi-controller SDN environment?

RQ3: What are the security issue mitigation approaches over multi-controller-based SDN?

1.4 Aim and Objectives

The major aim of this research is to focus on designing an efficient controller placement strategy along with a load balancing scheme to solve the MCPP in a SDN environment. The major objective(s) of this research are to:

1. optimally locate the multiple controllers in the SDN.

2. measure load statistics of multiple controllers and develop optimum load balancing strategy.

3. detect and mitigate the security attack over multi controller based SDN.

1.5 Scientific Contributions

The contributions of this research are the published archival/international journal listed in Table 1.1. Three journal papers (JP), one review paper (RP), and One conference paper (CP) were published during the period of this research study. Now, another journal paper is under review.

The published contents during this research are mostly adapted in this structured thesis with appropriate permissions obtained from the copyrighted article publishers.

1.6 Dissertation Organization

The thesis is organized into five chapters. The chapter-wise structuring of the contents is summarized here. Section 2 presents the literature review on controller placement implementation in SDN with a detailed concept of the NMR algorithm, traffic-driven load balancing, and SDN security. In Section 3, the methodology of our research work, including the experimental environment setup and dataset used for both MCPO and traffic load balancing, is discussed. Section 4 presents the overall analysis of my implementation approach and its comparison with others, while section 5 concludes the paper with possible future works.

- **Chapter 1** (This chapter) introduces the thesis related to MCPP and traffic controller load balancing in a multicontroller SDN environment, with a preliminary background of the research, statement of the problem, research questions, research aim, and objectives. A summary of research contributions is also listed in this chapter.

- **Chapter 2** provides a detailed literature review on SDN, SDN architecture, and CPP. A comparative analysis of different SDN controller placement strategies is presented. MCPP, scalability concerns in SDN, performance metrics in SDN, mainly latency and controller load balancing, traffic classification approaches for load balancing, SDN security with multi-controllers, and research gaps and the proposed approaches are discussed in this chapter.

- **Chapter 3** includes the research design and methodology. An overall theoretically separate framework for MCPO, traffic-driven load balancing, and attack detection and mitigation over an optimal multi-controller placement SDN environment is presented for analysis and evaluation of the research works. The different algorithms used during research activities are well mentioned.

- **Chapter 4** discusses the results and analysis of experimental works and the necessary interpretations.

- **Chapter 5** concludes the research study, highlights the limitations of the research, and possible future enhancements. It also summarises the contributions of scientific papers published.

Table 1.1: Peer reviewed archival journal papers

ID	Description	Publisher	Objective Met
JP1	Sapkota, B., Dawadi, B. R., & Joshi, S. R. (2022). Multi-controller placement optimization using naked mole-rat algorithm over software-defined networking environment. Journal of Computer Networks and Communications, 2022(1), 3145276.	Journal of Computer Networks and Communications - Wiley, USA	OB1
RP	Sapkota, B., Dawadi, B. R., & and Joshi, S. R. (2024). Controller placement problem during SDN deployment in the ISP/Telco networks: A survey. Engineering Reports, 6(2), e12801.	Engineering Reports - Wiley, USA	OB1, OB2
JP2	Sapkota, B., Dawadi, B. R., Joshi, S. R., & Karn, G. (2024). Traffic-Driven Controller-Load-Balancing over Multi-Controller Software-Defined Networking Environment. Network, 4(4), 523-544.	Network-MDPI, Switzerland	OB2
JP3	Sapkota, B., Ray, A., Yadav, M. K., Dawadi, B. R., & Joshi, S. R. (2025). Machine Learning-Based Attack Detection and Mitigation with Multi-Controller Placement Optimization over SDN Environment. Journal of Cybersecurity and Privacy, 5(1), 10.	Journal of Cybersecurity and Privacy - MDPI, Switzerland	OB1,OB3
CP1	Sapkota, B., Dawadi, B. R., Joshi, S. R., & Thokar, B. (2025, April). DDos Attack Detection and Mitigation Using Machine Learning in a Balanced Multi-Controller Software-Defined Networking. In 2025 International Conference on Inventive Computation Technologies (ICICT) (pp. 1725-1732).	IEEE	OB1, OB2, OB3
JP4	Reinforcement Learning-Based Multi-Controller Load Balancing in Software-Defined Networking *under review*	International Journal of Computer Network and Information Security	OB1, OB2, OB3

Chapter 2

Literature Review

In this chapter, a detailed literature review on the latest networking paradigms, viz., SDN and its architecture, and CPP, are carried out. Several SDN controller placement options are compared and contrasted. MCPP, scalability concerns, and performance metrics, including latency and controller load balancing, over SDN are presented. Traffic-driven load balancing is identified in a multi-controller environment. SDN security is highlighted briefly.

2.1 SDN in ISP/Telcos and Data Center Networks

SDN is a network management approach that makes use of programmable network configurations to enhance performance, lower the cost of network provisioning, and simplify network monitoring. Hence, SDN technology is a type of network management technique that enables a high level of programmability and centralized management. Seyedkolaei et al. in [31], SDN's core idea is to create programmable networks through the division of the control plane and data plane in order to enhance network operation. The data plane, often referred to as the forwarding plane, is in charge of directing traffic in accordance with the decisions made by the controllers. The control plane has the responsibility of managing traffic on the network and establishing guidelines and regulations for the data plane devices(s). It provides the data required for routing.

SDN virtualizes network services, reducing hardware needs and enabling software and programs to manage data centers. SDN offers the programmability of network elements and the creation of new routing and forwarding technologies without the need for hardware upgrades [32].

Kobo et al. [33] add that, currently, the SDN paradigm is being adopted and used

for a variety of computing models, including cloud, fog, and edge computing, as well as a variety of networking types, including corporate, mobile, and wireless. The SDN approach advances networking through innovation, simplification, and development. The SDN paradigm has shifted network intelligence from network devices to a central controller. Controllers are distributed out over a network to improve reliability, balance load, and eliminate single points of failure.

Some benefits of SDN may include the following: network management and visibility; reduced hardware footprint and opex; simplified policy changes; and networking innovations. There are still some challenges with SDN, nevertheless.

The SDN concept is being used and adopted by emerging architectures such as Network Function Virtualization (NFV), the Internet of Things (IoTs), data centers, vehicular ad hoc networks, cloud computing, as well as more traditional networking sectors like sensor networks, optical networks, mobile networks, and wireless networks [34].

Both scientific research and business are becoming more interested in SDN. With multiple companies, for example, Google, Facebook, Yahoo, and Microsoft, pushing and implementing SDN through open standards development, SDN has now had a significant impact on the industry. SDN is currently being used in a variety of network types, including enterprise and goverment networks, datacenter networks, wide-area networks (eg. Google B4), mobile networks and Internet exchange points etc. [35].

The SDN paradigm is used in data centers, cellular providers, service providers, businesses, and residences. The majority of vendors are accepting SDN in order to develop products and services that will satisfy the demands of leading network owners and operators and offer novel concepts to the market. For example, Google, Verizon, NTT Communications, Deutsche Telekom, Microsoft, and Yahoo! claim that SDN will enable them to create networks that are simpler, easier to design and program, and easier to manage [36]. Researchers used partners to deploy SDN technology over a three-year period at their campus and at a number of other campus sites across the country.

SDN deployment and implementation for a large-scale network, however, remain unresolved research questions. Multiple network vendors have released OpenFlow-capable switches as products and described their SDN plans. Also, many of the largest networking equipment manufacturers, including Ericsson, Cisco, VMware,

Huawe, NetApp, Alcatel-Lucent, HP, Cumulus Networks, Brocade and Juniper Networks, have already launched SDN projects. When choosing an SDN solution provider, companies must consider the following features: cloud compatibility, integration readiness, data and analytics, programmability, and support. Network programmability, logically centralizing intelligence and control, abstraction of the network, and openness are key areas in which SDN technology can benefit a business.

2.1.1 An overview of SDN architecture

The SDN architecture was developed in the Open Networking Foundation (ONF) Architecture working group as ONF, SDN Architecture, Technical Reference (TR-502), June 2014 June, 2014, and an updated SDN architecture published in February, 2016. The reference architecture for the generic SDN is shown in Fig 2.1 [11]. It is made up of three layers: a control layer in the middle, an infrastructure layer at the bottom, and an application layer on top.

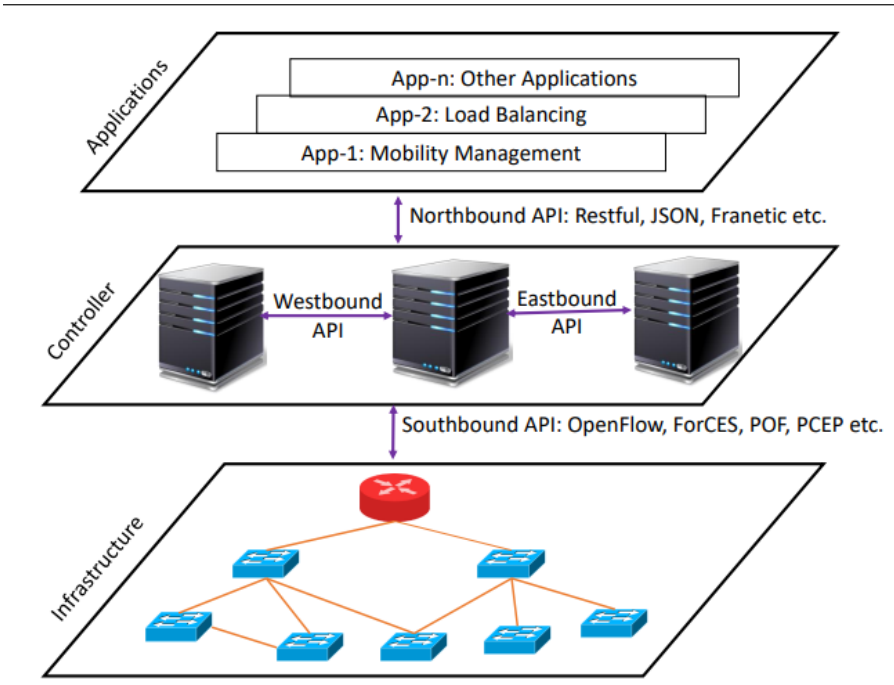


Figure 2.1: Generic layered view of SDN.

1. **Infrastructure layer**
   It is composed of forwarding components like switches, routers, openVswitch and wireless access points, among others. These devices are responsible for collecting and sending packets in accordance with the controller's instructions. Each switch keeps one or more flow tables, each of which has entries that consist of the entry identification, action, and statistics fields. When a packet

arrives, its header is compared to the entries in the flow table. If there is a match, the entry's stated action—such as forwarding the packet to a certain port or discarding it—is executed. If there is no match, the packet is sent to the controller for processing.

The switches are responsible for managing the known traffic going into the network from the hosts and maintaining the flow table using the OpenFlow protocols.This helps reduce the overload on the controller and allows it to perform tasks such as classification and controller shifting if required.

2. **Control layer**

The controller controls and programs forwarding devices through the southbound interface. This transmits the policies to the data plane and determines the routing of network traffic based on the requirements of the application. Also, this allows real-time manipulation of flow tables in network elements based on performance and service needs. In essence, the controller provides a centralized view of the network for better management and performance tuning and offers network abstraction for high-level network functionality.

For managing the infrastructure components, this consists of one or more controllers. These controllers are responsible for keeping the entire picture of the network using information collected by the forwarding devices and also for forwarding rules to switches. The primary functions of the controller are mostly related to topology and traffic flow management [37]. Link discovery, topology matnagement, decision-making, storage management, and flow management are the major components found inside the controller core. The controllers use a southbound interface, such as OpenFlow, for communicating to the forwarding devices. The east and west bound interfaces of the control layer are utilized for communication when there are multiple controllers as shown in Fig 2.1. The optimal selection of controllers is based on the controller features. Most popular open source SDN controllers in industry and academia includes the ONOS, ODL, OpenKilda, Ryu and Faucet [38].

The modularity of each controller is governed by the design focus and programming languages. The controller's portability and performance are affected by the programming language choice. For example, Python based controllers such as Ryu provides a well-defined API for developers to change the way components are managed and configured [39]. Shah et al. [40] insists that Java is the

best option, because it allows multithreading and is cross-platform. Python, however, has performance problems with multithreading. There are issues with memory management in C and C++. The .Net languages are runtime platform-dependent (Linux compatibility is not supported). Thus, authors in [40] demonstrate that among different controllers (NOX, POX, Maestro, Floodlight, Ryu), Java-based Beacon has the greatest performance.

Some of the examples of controllers are ONOS, OpenDaylight, Floodlight, Beacon, Ryu, and POX etc [41], [42]. Specifically, the southbound interface named OpenFlow allows for communication between the control layer and the network layer.

Hu et al. [43] explained the analysis done on the SDN controller based on throughput, response time, etc. To analyze the performance depending on variables like RTT, jitter, delay, etc., authors employed a mininet emulator that includes a Ryu controller. Some OF controllers are Summarized in Table 2.1.

Table 2.1: OpenFlow controllers

Ref.	OF Controllers	Programming Language
[33]	ONOS,ODL	Java
"	SNAC, Hyperflow	C++
"	Elasticon	-
[38]	POX, Ryu	Python
"	Maestro, Floodlight	Java
[44]	NOX, Runos	C++
"	Beacon, Iris	Java
"	MUL	C

3. **Application layer**

The SDN's control and data plane separation allows for effective domain-wide traffic routing and management. Programming forwarding devices is the responsibility of the controllers in the control plane, whereas network programming and policy enforcement are the responsibilities of the application plane, which is the top layer [2].

SDN applications use the controller's API to configure flows to route packets along the best route possible, load balance traffic, handle link failures, and divert traffic for auditing, authentication, inspection, and other similar security-related tasks. Thus, access control, firewalls, load balancing, network virtualization, security monitoring, etc. are a few of the SDN applications. Because they are event-driven, SDN applications' responses might be as simple as a default action or as complicated as a dynamic reaction.

Several end-user applications, including network monitors, time delays in traffic, traffic classifiers, and load balancing, are implemented in this layer. Network monitor keeps track of incoming traffic and counts the bytes, while traffic classifier uses machine learning to classify incoming traffic in real time, and load balancing reroutes incoming network traffic to the chosen controller based on the nature of the traffic.

4. **Open interfaces**

OpenFlow, ForCES, OpFlex, PCEP, and other southbound interfaces specify the communication protocol between data plane nodes and control plane components. Despite the fact that each has a different design objective, the command signals between the control plane and data plane are implemented using the SDN southbound protocols.

OF is the most commonly used Southbound Interface (SBI) and is a de facto standard for industry. The fundamental responsibility of OF is to define flows and classify network traffic based on a predefined rule. Northbound interfaces (NBIs) like REST, OSGi, FML, Procera, etc. provide an interface for application developers to create applications. Controllers support a number of NBIs, but most of them are based on REST APIs [45]. NBIs and SBIs affect controller placement decisions by influencing latency, scalability, reliability, and interoperability. To improve controller interoperability, standard east-west interfaces like ALTO, Hyperflow, etc. define several network controller communication protocols.

2.2 Controller Placement Problem

SDN is capable of resolving current network issues and enhancing network performance. Although there are other issues and opportunities for research in SDN, the

controller placement problem (CPP) is considered to be the main issue that could have a significant effect on network performance [46].

CPP primarily relies on figuring out how many controllers are needed overall and where they should be placed to improve network performance. Through regular message exchanges between controllers and switches, all SDN operations are completed. Therefore, message exchanges are significantly impacted by the location of the controller(s), which has an impact on SDN performance.

Kumari et al. [47] focus primarily on the CPP solution approaches. The paper offers a formal definition of CPP and an in-depth review of the different performance indicators and characteristics of the available CPP solutions.

In terms of CPP, Wang et al. [48] outlined a novel strategy to reduce propagation latency between SC, offered many lines of future study, highlighted some difficulties in the field, and opened relevant CPP questions. The need for an efficient algorithm, multi-objective optimization, cooperation among controllers, and cost awareness are a few of the issues described.

Chen et al. [49] added that the deployment model and performance optimization are the two key areas of interest in the CPP research. Numerous network performance measures, including propagation delay, controller capacity, node failure, deployment cost, and energy savings, have been taken into account from the perspective of performance optimization.

Abdel Rahman et al. [50] studied the best placement in the scenario when switches and controllers are linked via Wi-Fi networks. Hollinghurst et al. [51] presented a rigorous comparison of four widely used approaches to the controller placement problem. Ul Huque et al. [52] suggested a simple approach to satisfy the necessary SC latency in large-scale networks while dynamically minimizing the number of provided controllers. Kesentini et al. [53] used a method based on the Nash bargaining game to minimize SC delays, CC delays, and controller load balancing all at once.

Kobo et al. [33] thoroughly examined two issues: controller placement and controller mastership reelection. The researchers discussed and used the k-means model for the placement of local controllers and the re-optimized k-means or k-center model for the placement of global controllers.

In order to maximize network reliability, controller load balance, and latency between controllers and switches, Zhang et al. [54] formulate the multiobjective optimization controller placement problem (MOCP). They then determine the best locations for controller placement, the nodes that each controller can control within those locations, and the best way to distribute their routing requests among these controllers. The authors propose a mathematical model of this problem as an objective optimization function. The adaptive bacterial foraging optimization (ABFO) technique is designed in response to the computation complexity according to the actual network state in order to resolve this model.

In general, CPP can be widely divided into two classes: CCPP and UCPP.

1. **Capacitated CPP (CCPP)**

   When placing controllers in CCPP, load and capacity are taken into account. If load and capacity are not considered, there is a possibility that the controller will fail. Some controllers could be excessively loaded, while others might be hardly loaded.

2. **Uncapacitated CPP (UCPP)**

   Each SDN controller in UCPP has an unlimited capacity, and the load placed on the controllers is neglected.

Getting the optimum location and satisfying results when solving CPP is challenging and involves meticulous planning and wise decision-making. In order to appropriately arrange many controllers, it is important to take into account the various factors that affect SDN performance. These CPP objectives and challenging factors are presented in Table 2.2.

Table 2.2: Factors of CPP

Ref.	**CPP objectives**	**CPP challenges**
[5],[55],[56],[57], [58]	Latency, Control Load balancing, Flow setup time, Availability, Reliability, Resilience, Capacity, Energy, Cost, and Multi objectives	Scalability, Controller capacity, Fault tolerance, Interoperability, Packet flow rate, Flow set-up time, and load balance

Das et al. [57] describe the CPP in SDN and emphasize its importance. In addition to the CPP's main use cases in data center networks and wide area networks,

they also examine the CPP's most recent uses in a number of new fields, such as mobile/cellular networks, 5G, named data networks, wireless mesh networks, and VANETs. Regarding the CPP issues, they haven't paid enough attention to large ISP/Telco networks.

2.3 Comparative Analysis of Different SDN Controller Placement Strategies

Currently, a number of solutions have been developed and proposed to address SDN-based CPP. Existing solutions have been categorized in terms of controller allocation and the optimal number of controllers by minimizing network latency (propagation, controller processing), deployment cost, and energy consumption, as well as maximizing reliability and resilience [28].

The findings illustrate the existence of multiple CPP strategies, including cluster-based, LP/QP-based, evolutionary-based (bio-inspired, genetic algorithms), heuristic and greedy algorithm-based, simulated annealing-based, etc., while machine learning-based techniques are currently the subject of extensive research. Each of these strategies has a specific objective, a specific application, a certain weakness, and so on. A few different controller placement strategies for cluster-based large-scale networks are shown in the table 2.3 as examples.

Table 2.3: SDN controller placement strategies

Ref.	Algorithm/method	Topology	Metrics
[20]	K-Centre	Internet OS3E 256 topologies	SC latency
[59]	Clustering algorithm and greedy algorithm	Internet OS3E, Internet Topology Zoo	Reliability
[60]	Spectral clustering placement algorithm/k-self adaptive	OS3E topology	Latency, load balancing, Reliability
[61]	DBCP	OS3E topology	time consumption, propagation latency, and fault tolerance
[62]	Hierarchical K-means algorithm	Internet2 OS3E topology	latency and load balancing
[63]	Control Domain Division Algorithm (ILA-CDA) and Controller Placement Algorithm (PSO-CPA)	Internet2 OS3E topology	Propagation delay, link failure factors
[64]	Survivable Controller Placement	Internet2, RNP, and GEANT	Resilience and overload
[65]	RCP-DCP and RCP-DCR, modeled as mixed integer linear Programming (MILP)	different topologies available online in SNDlib database, Gurobi solver	Average control path length, expected control path loss, average connection availability and solving time
[66]	Clustering: PAM-B and Genetic: NSGA-II	topologies of different sizes, from 20 up to 1000 nodes	latency and loadbalancing

2.4 Multiple CPP

The single centralized controller cannot keep up with the expanding network and application demand. The most likely scenario is a single-point bottleneck and failure. Since the controller is solely responsible for all control activities, any network failure would have a substantial impact on network performance. As a result, the multi-controller is a feasible alternative for SDN in large-scale networks, and it matters for investigating SDN control plane scalability [67].

Hu et al. [43] concluded that it is ineffective to deploy just one controller to manage a large network. Placing multiple controllers is still challenging. In order to enhance the network's scalability and latency, it is most effective for big networks to use multiple controllers to manage the control plane traffic. There are two types of multi-controller architecture: flat architecture and hierarchical architecture.

Multiple controllers are deployed in a large-scale SDN environment in order to minimize latency, the workload of the controller, or multiple network performance indicators in relation to controller placement. Consequently, an SDN controller can manage multiple NOS [20]. In a small network, such as the network in a data center, a controller is adequate to manage the switches. In order to increase the scalability and stability of WANs, multi-controller deployment is becoming popular [68].

Dhar et al. [69] emphasize CPP issues with SDN and find that multiple controllers must be deployed to keep a large SDN scalable and reliable.

Holfed et al. [35] address new developments in architectural standards, protocols, and application designs to enable scalability in SDN. Selvi et al. [70] indicate that multiple controllers are used to manage and improve the reliability and performance of the network. The CPP for this setup is known as the multiple CPP (MCPP). The choice of metrics and the network topology themselves determine where to locate controllers.

Wang et al. [71] suggest a multi-controller architecture. In this work, researchers use a distributed strategy to evenly transfer the load among multiple controllers. The proposed model employs a distributed method and multiple controls to address the reliability issue. When a controller fails unexpectedly, the system can be restored by diverting all communications to other controllers that are still functioning normally.

Yuqi Fan et al. [72] present the RCP algorithm, which divides the network into

subnetworks and inserts a controller into each of them. Many controllers are efficiently positioned, and the connections between switches and controllers are defined using a local search technique. The simulation results show both the main and back systems' latency may be effectively reduced using the provided RCP technique.

A common trend in SDN is the addition of multiple controllers, which addresses reliability problems, horizontal expansion, and performance bottlenecks. The position and quantity of controller deployments, however, have been discovered to have a significant impact on network performance in a multi-controller network design as SDN networks have advanced. Therefore, determining the best location for the distributed control layer in the SDN network architecture remains a challenge.

2.5 Scalability Concerns in SDN

SDN-enabled network provides high performance, scalability, and availability, among other benefits. However, SDN suffers from a variety of performance and scalability issues, demanding major research efforts to maximize control plane scalability. Efficient resource allocation is in great demand and required for a scalable and adaptable network [73]. Multi-controller SDN frameworks like ONOS clusters, Kandoo, Hyoerflow, or Ravana are designed for scalibility, deployment and for other performance of network [74].

There are many different ways of determining scalability, including: administrative, functional, geographical, load, and generational. There is now a lot of work being done to improve the control plane's scalability in SDN. One of the most common control plane scaling approaches is topology-based scalability, which investigates the relationship between network topology (e.g., number of controllers, locations, etc.) and scalability concerns [75].

There are two types of topology-based controllers: centralized controllers and distributed controllers. Furthermore, the distribution has flat, hierarchical, and hybrid topologies [55]. The distributed and hierarchical SDN controllers can partition the network into smaller domains and assign some control functions to local or regional controllers. This can minimize the load and latency of the central controller while also improving the scalability and performance of the SDN network.

Domain partitioning concentrates on dividing the entire network into multiple SDN domains, whereas controller placement concentrates on locating the best places to

increase scalability. The scalability of multiple controllers is increased by using multiple controller deployments to divide a network into different domains [46].

Greater outcomes in terms of accessibility and scalability will be achieved by using multiple physically distributed but logically centered controllers across different network domains. However, this could lead to issues with load imbalances across several controllers. As a result, the Effective Initial Mapping in SDN (EIMSDN) method has been proposed to balance the load and reduce latency for managing the flow in the software-based network [76].

ISPs and data centers must constantly grow their operations to meet rising demand. ISP networks often feature fewer switches and routers than data center networks; nonetheless, nodes in such networks are typically geographically distributed. The huge diameter of these networks contributes to controller scaling issues, flow setup and state convergence latencies, and consistency requirements. Scaling while ensuring reliability and performance might be difficult [77].

ISPs struggle to continuously provide a high-quality service with the rise in connected devices and data-intensive applications. The ability of a system to accommodate an increase in usage, traffic, and data without significantly altering the infrastructure is referred to as scalable ISP/Telco infrastructure. The following are some challenges frequently encountered when creating and maintaining scalable ISP/Telco infrastructure: cloud dependencies, traffic peaks, data management, security, latency, cost, and maintenance.

ISPs can get a lot of benefits by implementing an efficient telecommunications traffic management system, including improved network performance, a better user experience, reduced latency, enhanced security, and better resource allocation.

2.6 Performance Metrics in SDN

Numerous qualitative performance measures and techniques have been offered in the literature to address the CPP in SDN. Propagation latency, reliability, load distribution, and failure resilience metrics are enhanced by effective controller placement. A few important SDN Performance metrics are mentioned in Table 2.4.

In this report, two performance metrics, latency (the primary performance parameter used in CPP) and controller load balancing, have been discussed. The bulk of research to date has found that network latency between controllers and switches

Table 2.4: SDN performance metrics

Ref.	Performance metrics
[5]	latency, Flow Setup time, Availability, Capacity, Energy and Cost
[78]	Latency, Degree of load balancing, Throughput, Utilization, Execution time, Peak load Ratio, Response time, Overhead, Root mean Square Error, Packet loss rate, Percentage of matched dead line flow, Energy consumption, Migration cost, Forwarding centric, Guaranteed bit rate, Overload ratio, average number of synchronization per minute, Workload and Cumulative frequency
[79]	Throughput, Fairness, QoS, Complexity, Congestion Control, Interface mitigation and Delay

and from controller to controller is the most relevant issue. Because it has a significant impact on SDN's overall performance. A proper placement of controllers should reduce latency, and each controller shouldn't be overloaded.

With an appropriate load-balancing strategy, response time and packet loss ratio could be efficiently decreased, resource usage could be improved, and overload could be avoided. Additionally, it might increase the network's capacity for expansion, reliability, packet delivery efficiency, and durability [80].

It is essential to have a common benchmarking methodology for evaluation because there are multiple controllers available with varying architectures and properties. Zhu et al. [81] provide a thorough review of benchmarking approaches and tools for SDN controllers. The software tool used for benchmarking SDN controllers must be exceedingly efficient and precise. The most popular benchmarking tools are CBench, PktBlaster, OFNet, HCprobe, OFCBenchmark, WCBench, and OFCProbe.

2.6.1 Latency

The amount of time it takes for data to move from one point to another between sender and receiver or between a specific user action and the response is known as latency. Common latency problems can be worth looking into. A low-latency network is one that has short transmission delays, which is desirable; a high-latency network, on the other hand, has greater transmission delays and is less desirable. In the context of WANs and SDN, reducing network device communication latency is a critical challenge that requires the optimal controller location. Javadpour et al. [82] highlighted that in SDN networks, the quantity of controllers and their placement can have an impact on two metrics: reliability and latency.

Since latency has an impact on ISP/Telco network performance, it is significant and

needs to be thoroughly studied. High-latency networks that experience long delays impede communication. It is true that we all desire communication with as little latency as possible. However, a network's typical latency varies slightly depending on the context, and latency problems change from network to network. Data packets are continuously processed and routed through various network channels made of wires, optical fiber cables, or wireless transmission mediums by network equipment including routers, modems, and switches. As a result, network operations are complex and several factors influence the rate at which data packets move. The following are common causes of network latency: medium of transmission, distance traveled by network traffic, network hops count, data amount, server performance, user issues, and Physical issues or inadequate hardware.

Latency or delay is one of the most commonly used performance indicators. Transmission, propagation, queueing, and processing delay make up the total latency. It is possible to evaluate latency between two nodes in one of two ways: switch to controller(SC) or controller to controller (CC) latency. In CPP, latency may be SS, SC, CC, or total latency of the network.

Heller et al. [20] begin a study on controller placement in SDN and suggest that propagation latency—both average and worst-case propagation latency—is the primary factor to be taken into account. This issue is conceptualized as a facility location issue, and K-center is used to solve it.

Selvi et al. [70] employ several qualitative metrics, including throughput, utilization, and latency. The amount of time needed to forward a packet through a network is called latency. There are several different types of latency, including communication latency and traffic delivery latency.

Wang et al. [83] identify that a critical difficulty in SDN is selecting suitable locations for controllers to reduce the CS latency. The CPP described a few of the performance factors that were taken into consideration, including control plane overhead, latency, load imbalance, cost, and connection. They use the CS latency (propagation, queuing, and processing delay) as a crucial performance parameter.

Mamushiane et al. [84] extended and used a facility location approach known as PAM with propagation latency to determine the optimum places to put SDN controllers. This study suggested using the Silhouette and Gap Statistics algorithms to decide how many controllers to deploy in a wide-area network. As a research study, they looked at the South African National Research Network (SANReN).

Rasol et al. [85] assesses the joint LRCP optimization model. With the help of alternate backup channels, LRCP gives network administrators a variety of options to balance the reliability and latency trade-offs between controllers and switches. This study suggests the CPL metric, the sum of average SC latency and average CC latency, to evaluate the controller placements offered by LRCP and determine how effective they are in an actual controller deployment.

In each link failure state, Fan et al. [63] further take into account the number of reroutings of the control path and the worst-case latency between the controller and the switch. To solve the problem, they offer a heuristic approach based on particle swarm optimization. The usefulness of the suggested algorithm is demonstrated by the numerical results. Additionally, it demonstrates that in the majority of link failure conditions, the suggested technique may ensure the latency and reliability of the control layer.

Yuqi Fan et al. [72] present the RCP algorithm. The objective of this approach is to to minimize the average latency between all switches and the appropriate controllers in the event of a single broken link. The latency of each path is made up of the latency of the primary path plus the average of any potential backup paths that might be available in the event of a single link failure.

Chen et al. [49] present that the network is separated into many sub-networks, and the essential performance metric is the latency between the controller and switch. The equation 2.1 by Liao et al. in [61] can be used to determine the latency model in this study.

$$L_{avg}(\phi(c)) = \frac{1}{|\phi(c)|} \sum_{s \in \phi(c)} d_{ijk}(s, c), c \in \phi(c) \tag{2.1}$$

where $d_{ijk}(s, c)$ is the Dijkstra shortest path distance, corresponds to the latency from the switch s to the associated controller c, $c \in \phi(c)$ denotes that controller c is placed at the position of one of the switches, $|\phi(c)|$ is the number of switches controlled by c.

Also, equation 2.2 and equation 2.3 for average and worst latency calculations are presented by Lu et al. [56].

$$L_{sc-avg} = \frac{1}{n} \sum_{s \in S,} d(s, c_i), s \in s_i \tag{2.2}$$

$$L_{sc-wst} = \max_{s \in S} d(s, c_i), s \in s_i \tag{2.3}$$

Where L_{SC-avg} and L_{SC-wst} are the average and the worst latency between switches and controllers and $d(s, c_i)|s \in s_i$ is the latency from the node s to the controller c_i of its subdomain. Similarly the average controller to controller latency can be formulated in equation 2.4 as,

$$L_{cc-avg} = \frac{1}{K} \sum_{c \in C, c' \in C} d(c, c'_i) \tag{2.4}$$

Where L_{cc-avg} is the average latency between the two controllers and K is the number of controllers.

Similarly, processing latency increases significantly as the controller's load exceeds or reaches its processing capability. As a result, the load on the controllers is often balanced in order to reduce processing latency ($L_{proc.}$). Equation 2.5 illustrates the mathematical expression.

$$L_{proc.} = min(\max_{c \in C} l(c) - \min_{c' \in C} l(c')) \tag{2.5}$$

In [4], discussion largely focuses on controller placement strategies that take into account optimization goals including latency, connectivity, cost, load, energy, QOS, and control plane overhead, or a combination of these objectives. A mathematical model for controller placement that reduces the worst-case latency in the event of controller failures was proposed by Killi and Rao [86]. Their model avoids a substantial increase in latency brought on by single connection failures. In the event of a single link failure, they also suggested a mathematical model to reduce the total worst-case latency and maximum worst-case latency.

For a reliable CPP, Singh et al. [87] suggest a VBO to guarantee that it reduces the overall average latency. Their results demonstrate that the proposed VBO algorithm outperforms other effective heuristic algorithms for the RCPP, such the PSO [88]. PSO and TLBO and their experimental results show that TLBO performs better than PSO for publicly accessible topologies [89].

Wang et al. [90] optimized the average latency and the worst latency utilizing the data plane traffic demands in their article, which concentrated on the CPP in the SDN multi-controller architecture. The traffic gravitation for the subdomain division problem was identified, and an enhanced LPA was created. On the other hand, using a heuristic approach based on open searching and the gravitational

force of nodes toward the controller, they determined the best controller placement position in each subdomain. Their experiment proved that the placement algorithm is more effective at minimizing the average latency and the worst latency with a fewer controllers.

Dhar et al. [91] proposed a mathematical approach called the "$-method" to form the clusters and put one controller in each cluster to reduce the worst-case SC latency. In terms of worst-case SC latency minimization with fewer controllers, the "$-method" outperforms other current techniques. They have also investigated the failure mode of a controller by assigning the switches from a failed controller to the controllers next to it, showing better performance in terms of network fault tolerance and resilience.

During the real-time migration of an existing legacy network into a SODIP6 network, the optimal path routing and BFR approach are used to determine the best location for the SDN control plane. In order to decide where to locate the controller using BFR, Dawadi et al. [92] analyzed the optimal latency.

In [83], the most important factor is the network latency between controllers and switches because it has a significant impact on the overall performance of the SDN. The end-to-end latency and the controller queuing latency are two more potential sources of latency that they research and examine in this work. A CNPA is then presented to partition the network in order to reduce end-to-end latency. The maximum end-to-end latency between controllers and switches can be reduced by each partition, according to the CNPA. The necessary number of controllers are subsequently installed in the subnetworks to further reduce the controllers' queueing delay.

An exemplary Problem formulation by Wang et al. [83] is described briefly. The Haversine formula in equation 2.6 is used to compute the great circle distances between pair of switches (or any two nodes) in Veness et al. [93] and Wang et al. [94] and the shortest path distance is calculated by using the Dijkstra's algorithm in Skiena et al. [95].

$$\text{Distance} = 2(r) \times arcsin(\sqrt{((sin^2(\tfrac{(\phi_2-\phi_1)}{2}) + Cos(\phi_1)Cos(\phi_2)sin^2(\tfrac{(\lambda_2-\lambda_1)}{2}))} \quad (2.6)$$

Where, ϕ is latitude of a node, λ is longitude of a node and r is radius of the earth. The end-to-end latency and queuing latency are calculated by equation 2.7 and equation 2.8, respectively.

End-to-end latency for a packet traveling from switch Sm to controller Cn is:

$$Le2e(S_m, C_n) = \sum_{i=1}^{h(S_m,C_n)} (\frac{P_i}{B_i} + \frac{d_i}{S} + D_{SPi}) \tag{2.7}$$

Here, packet transmission latency $(DT_i) = P_i/B_i$, where P_i is the amount of bits of a packet in link i and B_i is the bandwidth of the link. The propagation delay is given by $DP_i = d_i/S$, where d_i is the distance of link i and S is the signal speed at which data travels through the medium. The switch processing latency is denoted to D_{spi}, which is affected by the load of switch i.

Queuing latency:

$$L_{que} = \frac{L_q}{\lambda} = \frac{(m\rho)^m \times \rho p_0}{m!(1-\rho)^2\lambda} \tag{2.8}$$

Here m stands for the system's server/controller count. All controllers are thought to have the same service rate µ in order to make the analysis simpler. Then the traffic intensity is termed as ($\rho = \lambda$ /mµ).

Total Latency:

$$L_{total} = L_{e2e}(s_m, c_n) + L_{que} \tag{2.9}$$

Objective formulation:

$$min(maxL_{e2e}(S_m, C_n)) + L_{que} \tag{2.10}$$

such that $s_m, c_n \in SDN_i(\forall i \in k)$

Assuming the latency requirement is denoted by T_{th}, the maximum total latency is expected to be less than Tth.

Due to differing geographic distributions, each subnetwork has a significantly varying density of switches. The subnetworks with a lot of switches could have a lot of queuing latency. As a result, multiple controllers are installed inside subnetworks that contain a lot of switches.

2.6.2 SDN controller load balancing

Load balancing is a technique for allocating load to various network components or processors to maximize network performance and improve QoS [96]. Load balancing

makes it possible to predict bottlenecks before they occur.

SDN load-balancing techniques are more precise and perform better. Due to industrial concerns, load balancing is one of the most crucial topics in SDN-related research. Neghabi et al. [78] presented a thorough review of the load balancing strategies employed in SDNs, categorized into two types: non-deterministic and deterministic. A single switch architecture was used to illustrate and assess round robin, weighted round robin, least connections, weighted least connections, and random load balancing techniques [97].

To improve system efficiency and the user experience, the primary goal of load balancing is to prevent severe system load variance over extended periods of time. To effectively manage incoming traffic and resources and enhance network performance, the load balancing issue must be addressed due to the rising demand and depletion of resources. The role of the controller in SDN is to balance the load for better quality of service (QoS), which is one of the most crucial concerns [98].

The SDN controller plays an important role in enabling some of the load balancing objectives in distributed systems, including optimizing resource allocation, increasing throughput, reducing response time, and improving traffic [99]. Since the SDN controller has the potential to provide an extensive overview of the available resources, using multiple load-balancing techniques in SDN networks can improve network performance.

After reviewing the state of the art and research done on various controllers, load balancing can be divided into logically-centralized/physically-distributed. Physically distributed controllers are further subdivided into hierarchical, horizontal (flat), and virtualization controller load balancing.

Ahmad et al. [100] identified and emphasized the variety of SDN controllers and research challenges in different SDN control plane architectures, including distributed, hybrid, and centralized SDN control planes. They assessed the most well-known SDN controllers using the four performance metrics of scalability, consistency, reliability, and security. Issues like interoperability, consistency, network partition, controller placement and load balancing, and security have been raised in regard to distributed control plane architecture.

Ali et al. [101] proposed an efficient slave controller allocation-based load balancing technique called ESCALB in order to efficiently move switches to a controller with

idle resources in a multi-domain SDN-enabled IoT network.

Kumari et al. [102] and Li et al. [103] surveyed load balancing for both the data plane and the control plane, with a focus on SDN in the data center. Load balancing has been used to efficiently distribute network resources in order to improve both quality of service (QoS) and overall network performance, and this technology has proven essential.

In the review paper [24], we highlighted that it is necessary to measure the load statistics for multiple controllers by developing an intelligent load distribution strategy and optimize the number and location of controllers in order to establish balanced controller load distributions throughout the ISP/Telco network.

Alhilali et al. [2] classifies AI-based LB strategies into four main categories: nature-inspired techniques, machine learning, mathematical models, and other LB techniques. They provided comprehensive details on different metrics applied by several LB methods used to assess the performance of load balancing in SDN.

In the case of multicontroller SDN load balancing, a controller often experiences a higher load as the number of nodes linked to it increases. The network has additional delays as a result of queuing at the controller system caused by the rise in node-to-controller requests. Nodes of various controllers must be balanced in order to maintain resilience for controller placement.

Generally, SDN load balancing includes the following benefits as shown in Fig. 2.2 [104] .

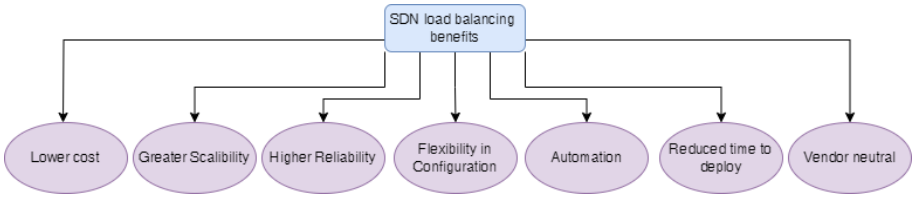


Figure 2.2: SDN load balancing benefits

Additionally, suitable load balancing helps in maximizing scalability, minimizing response time, maximizing throughput, minimizing resource consumption, avoiding overload of any single resource and so on. Neghabi et al. [78] point out that though load balancing techniques are important for SDN, there isn't yet a full and comprehensive systematic investigation in this field. They present a survey of the existing mechanisms and compare their properties, then describe specific common

load balancing techniques in the SDN and identify the types of difficulties that would be taken into consideration.

An SDN-based load balancer physically separates the network control plane from the forwarding plane. More than one device is able to be controlled at the same time when load balancing using SDN. There may be instances where some controllers are overloaded because of excessive traffic flow if the nodes are assigned to the nearest controller, utilizing latency as a metric or the shortest path distance between the node and controller. The number of nodes per controller in the network may be unbalanced [70].

In order to prevent overload on any of the resources, load balancing is a strategy that distributes the workload across multiple resources [23]. SDN load balancing techniques are more precise and perform better. Due to commercial concerns, load balancing is one of the significant topics in SDN-related research [105]. Dynamic load adjustment is a technique to distribute workload among current hosts by executing virtual machine migration and analyzing workload performance at various time intervals [106].

Using two strategies, controller clustering, which targets the dynamic controller, and switch migration, which manages the load distribution between controllers to maintain load, Hu et al. [43] present a complete review that explains the multi-controller concept and highlights the role of load balancing.

Using a new controller state synchronization approach and load variance-based synchronization (LVS), Zehua et al. [107] have addressed the looping and controller synchronization issues to improve load-balancing performance in multi-controller architecture.

By using a poly-stable matching algorithm, the authors Killi and Rao in [108] aimed to decrease the maximum load imbalance among controllers.

The controller capacity is impacted by increasing the number of SDN switches during migration. A suitable strategy is required to find the controller and install more controllers in the network for load balancing in order to handle the issue of increased control traffic with the rise of SDN switches [92].

An effective load balancing technique for numerous controllers in SDN is created by research by Zhong et al. [109]. A SCPLBS on SDN distributed controllers is presented. The time that controllers use to collect and publish loads can be changed

adaptively. If the loads on the controllers are higher than the threshold, the switch with the highest loads is transferred to the controller with the lowest loads.

The study in Zhou et al. [110] suggests a DALB. The aim of this algorithm is to balance the load on each controller. The switch with the highest load is relocated if its loads exceed the threshold. DALB reduces controller overhead by using a threshold for load collection. Adaptive threshold adjustments can be made for each controller based on the loads that are handled by it.

According to Konglar et al. [111], a distributed controller architecture, however, does not ensure that the overload issue can be fully resolved. In order to reduce the loads on the overloaded controllers, this work suggests a LDOP. They propose a controller manager that is only responsible for transferring switches from an overloaded controller to other controllers. The LDOP method can also lower the overall number of loads from all controllers that exceed the threshold. Switches are moved based on preventing switch migration from making a chosen controller an overloaded controller. The LDOP algorithm can operate when all controllers are overloaded.

In their research, Babbar et al. [112] have presented a latency-based load balancing solution for multiple SDN controllers. Their suggested technique resolves the load-balancing issues with multiple overloaded controllers in the SDN control plane by determining the necessary latency and addressing multiple overloads concurrently. Along with the migration, their algorithm's latency has decreased by 25% when compared to other algorithms.

In [113], the communication with the switches is handled by multiple controllers in a distributed architecture. As a result, each controller experiences a reduction in processing load as a result of the distribution of control traffic between switches and controllers, which has a positive load balancing effect.

The load balancing approach for distributed controllers in SDN presented in [114] is dynamic and adaptable and is based on a hierarchical control plane. They have put forward a useful load balancing mechanism for distributed controllers that may dynamically migrate the load via switch migration from the heavily loaded controller to the lighter loaded one in accordance with the load distribution of the control plane. According to the simulation results, the suggested technique can dynamically balance the load on the control plane and boost the throughput of distributed controllers.

Wang et al. [71] presented a load adjustment technique to be applied to each controller in order to achieve load balance among multiple controllers. In SDN systems, Three logical parts make up the suggested mechanism: a load collector, load balancer, and switch migrater.The experiment results demonstrated the effectiveness of the data gathering process and the adjustment of global and local loading. To simulate different loads for each switch, Cbench was used to generate traffic.

From their simulation results, Lin et al. [115] concluded that the proposed RCP-ILP enhances the robustness, the efficiency and load balancing of SDNs at the cost of least controllers placement against links failure.

Yang et al. [116] presented Simulated Annealing Partition-based K-Means (SAPKM), a low-complexity controller placement algorithm for SDWAN, to provide load balancing among distributed controllers. Through the simultaneous improvement of propagation delay and network reliability performance, experimental findings showed the efficacy of SAPKM in lowering average load and load balancing indices.

For a typical ISP, Al et al. [117] describe a SDN load-distribution method. To do this, researchers have created an optimization technique using linear programming. For this, a multi-objective model has been created. Among the numerous objectives are to increase link availability, maximize link utilization, and minimize the typical round-trip time for the most popular websites.

2.7 Traffic Classification for Load Balancing

Network traffic classification is the process of identifying and classifying the network protocols or applications that are present in a network. Network management, service measures, network architecture, security monitoring, and advertising have all made extensive use of traffic classification. Knowing what kinds of network applications flow through a network is a crucial challenge for Telco/ISPs.

Currently, network traffic classification is a hot topic in computer science. Understanding the variety of network applications used on a network is a crucial task for ISPs. Through this technique, ISPs or network operators can manage the overall performance of a network. Traditional techniques for classifying internet traffic include those that use ports, payloads, and machine learning. The machine learning (ML) approach is currently the most widely utilized method [118].

Machine learning-based approach [119]

The use of traffic classification techniques based on ML has recently solved the drawbacks of traditional traffic classification methods. In this method, a machine learning classifier is trained as input, and then unknown classes are identified using the trained sample prediction.

ML is the area of AI concerned with creating computer programs that can solve problems requiring intelligence by learning from data. There are three main branches of ML: supervised, unsupervised, and reinforcement learning.

I. Supervised learning:

In supervised learning (SL), the machine learns from training data. The training data consists of a labeled pair of inputs and outputs. So, the model (agent) is trained using the training data in such a way that the model can generalize its learning to new unseen data. It is called supervised learning because the training data acts as a supervisor, since it has a labeled pair of inputs and outputs, and it guides the model in learning the given task.

II. Unsupervised learning:

In unsupervised learning (UL), the model (agent) is trained based on the training data. But in the case of unsupervised learning, the training data does not contain any labels; that is, it consists of only inputs and not outputs. The goal of unsupervised learning is to determine hidden patterns in the input. There is a common misconception that RL is a kind of unsupervised learning, but it is not. In UL, the model learns the hidden structure, whereas, in RL, the model learns by maximizing the reward.

III. Reinforcement learning[120] :

Reinforcement learning (RL) is the task of learning through trial and error. Unlike other ML paradigms, such as SL and UL, RL works in a trial-and-error fashion by interacting with its environment. In this type of task, no human labels data, and no human collects or explicitly designs the collection of data. The goal in RL is to act. A powerful recent approach to ML, called deep learning (DL), involves using multi-layered non-linear function approximation, typically neural networks. DL isn't a separate branch of ML but a collection of techniques and methods for using neural networks to solve ML tasks, whether they be SL, UL, or RL. DRL is simply the use

of DL to solve RL tasks.

Traffic driven load balancing

Shafiq et al. [121] discuss network traffic classification approaches step by step. The classifiers Support Vector Machine, C4.5 Decision Tree, Naive Bays, and Bayes Net are used. According to experimental data, C4.5 classifiers perform quite well in terms of accuracy when compared to other classifiers.

Qazi et al. [122] introduce the framework Atlas, which uses ML-based classification techniques in order to incorporate fine-grained application awareness in SDN. The proposed approach can classify network traffic in real time since it gathers the packet size of the first N packets of a network flow. Atlas is installed on a wireless network and uses the Android OS to collect network traffic.

Using the distributed controllers design, Gasmelseed et al. [25], propose a new load balancing method for SDN that splits the traffic into TCP and UDP and employs a failover strategy to prevent the single point of failure problem, achieve fault tolerance, and provide a high-availability environment that ensures reliability.

According to this study, the suggested algorithm outperforms random, round robin, and weighted round robin in terms of availability, response time, transaction rate, throughput, concurrency, and packet loss. The algorithm also demonstrates its effectiveness in a dynamic network with heterogeneous traffic that demands high availability, low response times, high transaction rates, high throughput, minimal concurrency, and minimal packet loss.

Amaral et al. [123] use three supervised classification techniques, such as Random Forests (RF), Stochastic Gradient Boosting (SGB), and Extreme Gradient Boosting (EGB)-XG Boost. They set up this platform in an enterprise network and gathered data using the following schemes: Bittorrent, Dropbox, Facebook, HTTP, LinkedIn, Skype, Vimeo, and YouTube. The OpenFlow protocol was used to collect the data for this study, and features such as packet size, interval arrival time, source and destination IP/MAC/Port, flow length, byte count, and packet count of the first five packets were collected.

Raikar et al. [124] suggest a supervised learning model for classifying data traffic in the SDN context. SVM, closest centroid, and Nave Bayes are the three different models employed to categorize the data traffic based on the applications on a SDN

platform. In all three supervised learning models, the accuracy of traffic classification is greater than 90%.

Amiri et al. [125] describe a technique for raising the quality of service (QoS) for traffic flows in a cloud data center by utilizing SDN and ML-based traffic classification. The proposed real-time traffic classification module makes use of ML techniques to improve traffic classification accuracy. Compared to Equal Cost Multi-Path (ECMP), the proposed optimization method increases bandwidth consumption. This improvement may be extremely important in ensuring that users' interactions with cloud data centers have a sufficient QoE.

In order to address the mislabeled training data issue, Wang et al. [126] introduce a novel approach in their research called Noise-resistant Statistical Traffic Classification (NSTC), which integrates reliability estimates and noise reduction into traffic classification. In the case of large amounts of unclean data, NSTC can dramatically outperform state-of-the-art approaches in terms of classification performance.

Eom et al. [127] propose an SDN-based system for classifying network traffic. Applying four ensemble algorithms—Random Forest (RF), Gradient Boosting Machine (GBM), Extreme Gradient Boosting (XGBoost), and Light Gradient Boosting Machine (LightGBM)—researchers examine the classification performance of each algorithm in terms of accuracy, precision, recall, F1-score, training time, and classification time. The experimental results show that classifiers based on ensemble models outperform those based on the proposed framework and the real-world network traffic dataset. Notably, the classification performance achieved by the LightGBM model is the best.

Ejaz et al. in [128] balance the load with a focus on the controller. The incoming traffic is balanced using virtual SDN (vSDN) controller duplication and shared load. Using two vSDN controllers in a Mininet emulator, the authors experimentally confirmed the load balancing in the Fat-Tree architecture.

Hai et al. [129] provide load balancing in the control plane based on the load status and the dynamic weight coefficient of each controller to improve resource use, reliability and resilience in the network. Additionally, compared to current methods, communication overhead is greatly decreased by employing a pre-defined load threshold.

Mousa et al.[130] give an in-depth review of recently introduced SDN-based load

balancing and routing methods. There are different types of possible load balancing techniques, such as switch migration, routing, controller placement, traffic classification, heuristic algorithms, AI-based, etc.

In the published research paper [131], a novel population-based meta-heuristic method, NMR Algorithm, is proposed to position controllers on the site more effectively based on SC, CC, and load-balancing among the controllers. Two widely accessible standard topologies, Ernet and Savvis, are used to demonstrate the concepts and methods. When compared to the Bat method, the NMR algorithm's controller localization approach yields somewhat superior results.

Xue et al. [132] proposes a novel approach for load balancing of SDN which combines Genetic-Ant (GA) with Colony Optimization (ACO), called (G-ACO). It benefits from the speedy genetic algorithm (GA) global search and the effective search for an ACO optimal solution. Computer simulation reveals that the proposed scheme significantly improves the round-trip time (RTT) and packet delivery ratio compared to the round-robin (RR) and ACO algorithm by effectively achieving the LB.

Mohammed et al. [133] look at existing deep learning (DL) techniques to classify and predict traffic in an SDN environment.

Malik et al. [134] offer Deep-SDN, a new deep learning model for SDN that can quickly and reliably identify a variety of traffic applications. According to their research, the proposed model could achieve an overall accuracy of 96%. The researcher's claim is that the proposed model can accurately and quickly identify the different network traffic application types, making it useful for identifying online traffic.

Babayigit et al. [135] propose that the DL approach is highly efficient for SDN-based DCN load balancing compared to other algorithms such as an artificial neural network (ANN), SVM, and LR.

2.8 SDN Security with Multicontrollers

SDN is becoming more and more problematic as a result of the significant rise in internet users and the resulting increase in network security risks. One of the most common and hazardous types of attacks in SDNs is distributed denial of service (DDoS) [136], [137]. The network is divided into data plane, control plane, and application plane in SDN architecture. Depending on the target plane, DDoS attacks

in SDN can also be divided into three categories: application-layer, control-layer, and data-layer threats [138]. A number of strategies are employed for the detection, prevention, and mitigation of DDoS attacks, particularly on controllers in the SDN [139].

While SDN offers numerous benefits over cloud and network service providers (e.g., 5G/Telco/ISP), its widespread adoption is contingent upon addressing its security challenges. Targeted attacks have the ability to interfere with all three layers, primarily on the SDN software controller, making it the weakest link in the system overall, on the link between controllers (C2C), on the link between the controller and the switches (C2S), on the switches themselves, on applications, and on the link between the applications and the controller [140].

DDoS attack has the ability to completely destroy the network in addition to preventing unauthorized users from accessing and using network resources [141]. Consequently, defending the SDN network against DDoS attacks is essential.

There are different kinds of DDoS attacks, including ICMP flood, TCP flood , and UDP flood [142]. By running the attack source, an attacker drastically decreases the available bandwidth and stops the target host from communicating with the outside world by sending a lot of trash traffic to the target network.

Singh et al. [143] comprehensively reviewed about 70 well-known DDoS detection and mitigation techniques over SDN. Information theory-based approaches, machine learning- based methods, Artificial Neural Networks (ANN)-based methods, and other miscellaneous methods are the four groups into which these techniques are classified. All of the currently available DDoS detection solutions are divided into four groups based on the type of detection metric and detection technique used: information theory metrics, machine learning, artificial neural networks, and other defense solutions [138].

ML has become a popular tool for helping with threat and security analysis, instruction detection, and other related tasks. Deeper insights can be gained by correctly examining ML's support for a range of datasets produced by network traffic and a number of data characteristics [144].

Furthermore, ML has emerged as one of the most widely utilized methods for identifying DDoS attacks in recent years. Tahirou et al. [139] apply supervised machine

learning methods like Naive Bayes (NB), Support Vector Machine (SVM), and K-Nearest Neighbors (K-NN) in this context. The Canadian Institute for Cybersecurity (CIC)-2019 dataset is used to train these algorithms. Ultimately, a real time SDN simulation environment consisting of Mininet, Ryu controller, and SNORT NIDS is used to evaluate the performance of these approaches. As soon as an attack is detected and put to a blacklist for mitigation, the attacker's IP address and port number are blocked.

SVM, Hidden Markov Model (HMM), Decision Tree (J48), Naive Bayes, Advanced Support Vector Machine(ASVM), Logistic Regression (LR), Random Trees, Binary Bat algorithm, Random Forest (RF), and K-nearest neighbor (KNN) are some of the effective machine learning algorithms that many researchers have used for DDoS attack detection. Also, Many researchers use effective algorithms, such as Self-Organizing Maps (SOM), exact-STORM, Back Propagation Neural Network (BPNN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long Short Term Memory (LSTM), to identify DDoS attacks in SDN [138], [145], [146], [147], [148], [149], [150].

Meti et al. [151] use the NN classifier and the SVM classifier to identify the potentially hazardous connections. Mohs et al. [150] look at four machine learning methods—RF, KNN, Naive Bayes, and LR—for detecting DDoS attacks..

Current studies focusing on multi-controller based attack detection are limited in scope and do not address network optimization. Valizadeh et al. [152] and Gebremeskel et al. [153] proposed solutions for detecting DDoS attacks in multi-controller environments. The research conducted by Pandikumar et al. [154] effectively detects both inter-domain and intra-domain attacks but lacks an efficient mitigation module. The mitigation procedure they propose only reduces the intensity of the attack by dropping malicious flows and provides no communication mechanism for attack source blocking. This approach necessitates manual intervention for the permanent elimination of ongoing attacks. Similarly, the work by UmaMaheswari Gurusamy et al. [155] is limited to scenarios with tree-like topologies, allowing only one connection per controller. Additionally, none of these research focus on optimal number and placement of controllers, exploiting optimal controller placement would have resulted in better performance in terms of congestion, scalability, and speed (faster detection and mitigation).

Thus, majority of the existing literature dealing with threats in SDN accounts only for single controller deployed in an arbitrary network topology, which usually delivers

unrealistic or biased performance metrics like speed, throughput, and bandwidth. In addition, the mitigation module has mostly implemented without first obtaining the optimal placement of network elements, which prevents the network from obtaining the best possible bandwidth and speed. Existing works mostly have implemented an attack detection action by the controller; this approach keeps the controller busy and occupied. For ML training, CIC datasets have been widely used, which come with high processing and packet capture overhead.

2.9 Research Gap and the Proposed Approach

From the review of the literature, it is clearly seen that MCPO and an intelligent approach to load balancing in the SDN environment for the data center and ISP/Telco network are inevitable.

Numerical measures such as latency, load imbalance, energy, computation time, cost, connectivity, and control plane overhead, among others, have been presented in the literature to address the CPP in SDN. SC and CC latency play the most important roles among these measures since they have a significant impact on SDN's overall performance [156]. Therefore, multi-controller placement optimization in large networks, which is still a research gap, has been addressed by using NMR the algorithm by assessing latency and identifying solutions to reduce it in this study.

Although Nature-inspired algorithms may have certain drawbacks, more and more optimization issues are now being solved using nature-inspired algorithms to find the best solution. Additionally, because of their simple conceptual model and little requirement of gradient data, they are also easy to construct. The selection of NMR algorithms to solve the CPP in SDN is hence highly motivated. Therefore, in this report, the NMR algorithm, a novel meta-heuristic swarm intelligence algorithm, has been implemented to solve the optimization problem of controller placement by dividing its population into two distinct groups as worker (exploration) and breeder (exploitation) groups and controlling the transition between these phases. The NMR algorithm from references shows its superiority result while there are many other meta-heuristics like WOA, ALO and others GA, PSO, SA and GWO, each provides an adaptive trade-off for maintaining the balance in a complex search.

Moreover, in addition to the MCPO issue, the traffic classification-based load balancing performed by a strategically placed SDN controller is proposed and discussed. Using a broad perspective of the network, a network controller in SDN decides how

to forward data. As a result, it may manage network traffic "on demand" to meet application requirements.

Traffic management is essential for ensuring the quality of service (QoS) for real-time business traffic, which is sensitive to packet losses and delays. Therefore, a lot of researchers have concentrated on designing dynamic data forwarding path planning algorithms that may satisfy end-user requirements.

Therefore, the primary goal of the research for multi-controller SDN is to create a system that dynamically classifies network traffic and reroutes it according to the criticality of incoming data. With regard to traffic classification and controller load balancing, the development of a trained model is proposed, and its performance is measured with different ML algorithms in a multi controller SDN environment.

Additionally, this research work introduces the prevalent issues of MCPO in real life network topologies and load balancing for a SDN environment. Here, K-means++ and OPTICS algorithms are proposed to determine the optimal number and positions for controllers. Furthermore, as a security issue, the DDoS attack detection and mitigation process is proposed for an optimized multicontroller-based SDN network. Attack detection process is handled by the central IDS which is independent of controller, saving precious resources of controller for other tasks. Additionally, this work incorporates the use of primary dataset that produces more authentic representation of SDN network behaviour with no packet capturing and lower computational overhead.

2.10 Chapter Summary

A detailed literature review on SDN, CPP strategies, MCPP, scalability concerns in SDN, and performance metrics, including latency and load balance, has been carried out. Different traffic controller approaches are identified, and traffic-driven controller load balance is performed. Also, SDN security with multicontrollers is briefly presented.

Chapter 3

Methodology

The present research work's objectives are focused on optimum multi-controller placement problems, finding suitable solutions for MCPO, and optimum load balancing among the controllers. Similarly, this chapter covers the overall research design, including theoretical formulations about the MCPO and traffic load balancing models over SDN, problem formulation, different research algorithms for MCPO, traffic load balancing in research studies, experimental tools and techniques used, and data set generation. Additionally, optimal controller placement implementation, and machine learning-based DDoS attack detection and mitigation techniques in a multi-controller SDN environment are carried out.

3.1 Overall Design and Framework

The overall conceptual research framework is presented in Figure 3.1. The overall research was designed in the modules contributing to journal papers (JPs) and review paper (RP) to address the research questions outlined and as per the objectives defined.

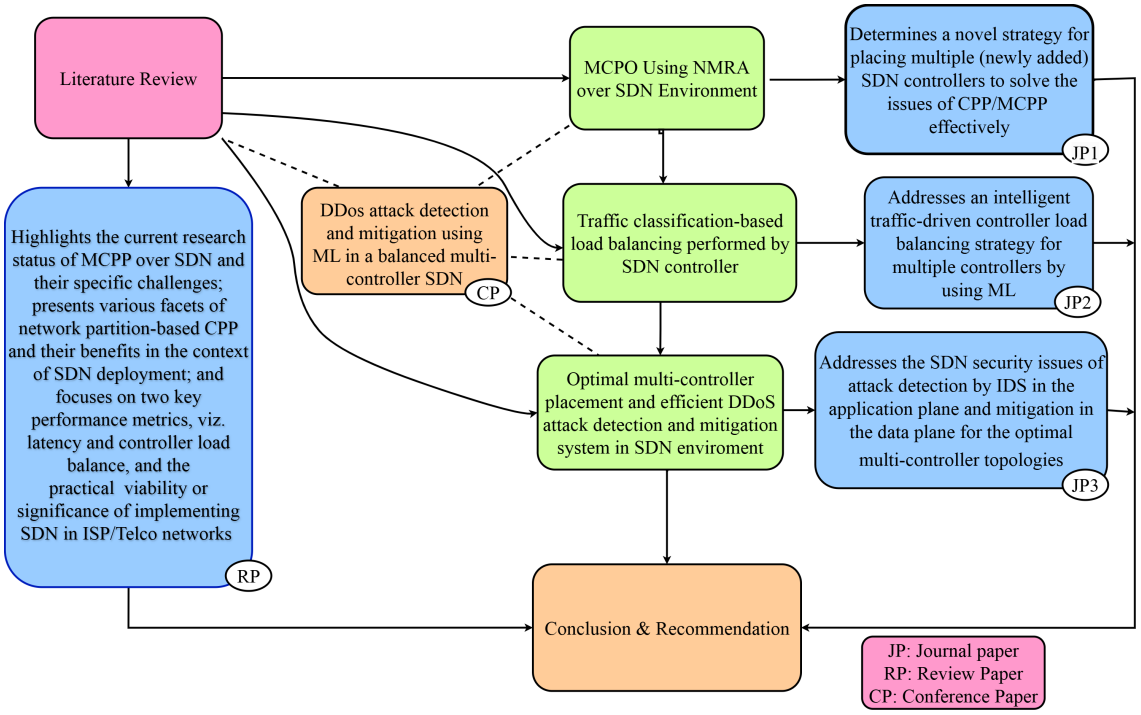


Figure 3.1: Overall research design framework

The details of the theoretical formulations, network topology development, data samples used, system scenario, different proposed algorithms, and necessary assumptions made in the simulation/emulation works are presented in the following sections.

3.2 Multi-controller Placement Optimization: Theoretical Formulations

3.2.1 MCPO using naked mole rat algorithm over SDN environment

NMR algorithm [22] is a stochastic optimization algorithm. The social behavior of the NMR imitates the mating pattern of these animals. The key features are summarized below.

1. NMR, a eusocial animal, can have upto 295 members with average number of members being 70-80.

2. The group is led by a female queen and the entire population is divided into breeders and workers. The most efficient NMRs among working groups are

breeders and are intended for mating with the queen.

3. All other necessary tasks are performed by the workers and the most efficient worker will be promoted to breeder's group. Thus, the workers who outperform the other workers will become breeders while the breeders who perform the worst will be shifted back to the worker's pool.

4. Ultimately, only the best breeder among the breeders will be the queen's mating partner.

These above mentioned rules formularize the concept of NMRA. The population of NMR is initialized during the first phase. Then the NMRA population is divided into workers and breeders. The selection of the breeders is based on their breeding probability. The algorithm can be described in steps listed below.

1. **Initialization:** The population of n NMRs are randomly generated in the range of $[1, 2, ..., n]$. Each NMR is represented in D-dimensional vector space. Here, D represents the number of variables or parameters to be tested in the problem. Each NMR is initialized as give in Equation 3.1.

$$NMR_{ij} = NMR_{min.j} + U(0,1) \times (NMR_{min.j} - NMR_{max.j}) \tag{3.1}$$

Where $i \in [1, 2, ..., n]$, $j \in [1, 2, ..., d]$, $NMR_{i.j}$ is the i^{th} solution in the j^{th} dimension, $NMR_{min.j}$, $NMR_{max.j}$ are the lower and upper bounds of the problem function respectively and $U(0, 1)$ is the uniformly distributed random number. The objective function is evaluated and its fitness is estimated after it has been initialized. b breeders and w workers are determined based on fitness, and the overall initial best solution d is calculated. After initialization, the NMR population is subjected to multiple cycles or iterations of the worker and breeder phases of the search process.

2. **Worker phase:** During this phase, the workers are inclined to improve fitness to gain opportunity to become a breeder and eventually mate with the queen. The new solution based on its previous experience and its information available locally. Old solution is memorized unless the mating fitness is better than the previous one. The best fitness of worker will be remembered after the completion of search process. A new solution is produced from old solution

given by Equation 3.2:

$$w_i^{t+1} = w_i^t + \lambda(w_j^t - w_k^t) \tag{3.2}$$

Where w_i^t corresponds to the i^{th} worker in the t^{th} iteration, w_i^{t+1} is the new solution or the worker, λ is the mating factor and w_j^t and w_k^t are two random solution chosen from the worker's pool. The value of λ is obtained from the uniform distribution in the range of [0,1].

3. **Breeder phase:** In breeder phase, the breeder NMR updates the fitness to retain its position as a breeder and improve its chances of mating. All the breeders updated from the workers phase are based on their breeding probability (BP) with the best fitness as their baseline. The value of BP lies in the range [0,1]. If a breeder can't update its fitness, as its breeding probability is low, then it could be pushed back to worker category. The positional update is based on Equation 3.3.

$$b_i^{t+1} = (1 - \lambda)b_i^t + \lambda(X_{best} - b_i^t) \tag{3.3}$$

Here, b_i^t corresponds to the breeder i in the iteration t, λ factor controls the mating frequency of breeders and helps in identifying a new breeder b_i^{t+1} in the next iteration, and $(X_{best})is \leftarrow$ NMR with the best minimum latency.

The concept of mating pattern of breeders with the queen is used for finding the appropriate solution of the considered problem. On the basis of the idea that the best workers can be promoted to the breeders and the best breeders drift towards mating the queen is followed to devise a new solution. The algorithm focuses on determining the best breeder which can mate with queen and thus devising the potential best solution of the problem. Breeding probability controls the shifting of worker phase towards the breeding phase [157].

The appropriate generic diagram to support this section is shown in Figure 3.2.

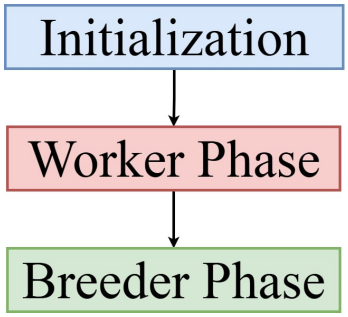


Figure 3.2: Generic diagram of Naked-mole Rat phases

3.2.2 Problem formulation

A network topology is represented by an undirected graph $G(S, E)$, where S represents the set of switches and E represents the set of bidirectional physical links interconnecting the switches. Table 3.1 provides the set of notations to represent sets, decision variables and constraints. All the quations are referred from [22].

Table 3.1: Notations and descriptions

Notations	Description
S	A set of switches present in the network topology, $S = \{s_1, s_2, s_3, ..., s_n\}$, where n is the total number of switches. $L(s)$ is the load associated with the switch.
C	A set of controllers to be installed in the network, $C = \{c_1, c_2, .., c_k\}$, where k is the number of controllers to be installed. $L(c)$ is the total load capacity associated with the controller.
P	The set of all possible locations for controllers' placement.
$T(c)$	The set of forwarding switches connected by controller.
$d(s, c)$	A function that calculates the shortest path between a switch $s \in S$ to one of the controller $c \in C$.
$d(c_i, c_j)$	A function that calculates the shortest path from controller c_i and c_j.
x_{cp}	A decision variable that is: $= \begin{cases} 1 & \text{if } c \in C \text{ is installed on location } p \in P \\ 0 & \text{otherwise} \end{cases}$

Average SC latency is the most commonly used metric in CPP. It calculates the average distance between the placed controllers' location and the switches assigned to them. It reflects the basic performance of propagation latency in the SDN by representing the average value of packet transmission latency between the switch

and the controller. The mathematical expression can be expressed in Equation 3.4.

$$L_{sc-avg} = \frac{1}{n} \sum_{s \in S, c \in C} min[d(s,c)] \tag{3.4}$$

The sub-network created among the controllers' is located on the controller plane, thus the latencies among these controllers is also critical aspect of SDN. Communications between the controllers are crucial for achieving the consistent view of the network's state which is utilized by network application for proper operation. The communication overhead maintained by the shared state among the controller is very significant. The controller to controller latency is mathematically represented by Equation 3.5.

$$L_{cc-avg} = \frac{1}{k} \sum_{c \in C, c' \in C} min[d(c,c')] \tag{3.5}$$

It is considered that all the switches are connected to the closest controller based on latency as the metric, unless the addition of the switch exceeds the controller capacity, i.e., the controller already has the maximum number of switches it can handle. In such a scenario, the switch is connected to the second nearest controller. This can be mathematically represented in Equation 3.6.

$$\sum_{s \in T(c)} L(s) \leq L(c) \qquad \forall c \in C \tag{3.6}$$

For the placements of C controllers, the global average latency can be represented by Equation 3.7.

$$G(C) = w_1 \times \frac{1}{n} \sum_{s \in S, c \in C} min[d(s,c)] + w_2 \times \frac{1}{k} \sum_{c \in C, c' \in C} min[d(c_i, c_j)] \tag{3.7}$$

Where, w_1 and w_2 are weights, such that $w_1 + w_2 = 1$.

The objective function of capacitated controller placement problem for global average latency is given by constraint Equation 3.8.

$$\text{Minimize } G(C), \text{ subject to} \sum_{s \in T(c)} L(s) \leq L(C) \forall c \in C \tag{3.8}$$

For k number of controllers to be deployed in an SDN, the goal is to optimize the placement of controllers such that the value of $G(C)$ is minimized subjected to the

constraint presented in Equation 3.8 and $|C| = k$.

The great circle distance between the pairs of switch is computed using Haversine distance approach. The shortest distance between two points on a sphere, measured along the earth surface is known as great circle distance. Alternative to this approach is Law of cosines which is actually accurate only for shorter distance. The Equation 3.9 is used to compute the great circle distance as defined by Haversine approach, where ψ_1 and ψ_2 represent the latitudes of the point 1 and point 2, λ_1 and λ_2 represent the longitude of point 1 and 2 respectively and r is the radius of the earth at a constant 6371 km.

$$\text{distance} = 2(r)arcsin\sqrt{sin^2(\tfrac{\psi_2-\psi_1}{2}) + cos(\psi_1)cos(\psi_2)sin^2(\tfrac{\lambda_2-\lambda_1}{2})} \tag{3.9}$$

The mapping of the switch to controller is the relationship of how the switches are controlled by the controller. The assumption is that one switch can be controlled by only one controller and is based on the shortest distance between the switch and the controller. Also, the switch positions where the controllers are placed are controlled by default by the deployed controller. The selection of the second nearest controller is only deemed necessary in situation when the load of the controller exceeds the switch handling capacity.

Dataset

The dataset used in this study are the real topologies that are obtained from the Internet Topology Zoo (`http://www.topology-zoo.org/`) and the standard topology e.g. 'Savvis' and 'Ernet'. Savvis is the network topology representing the backbone network of USA, which consists of 19 nodes and 20 edges connection between them. 'Ernet' is the backbone network of India which consists of 16 nodes and 18 edges.

3.2.3 Naked mole-rat controller placement problem

NMR algorithm is used for optimizing the controllers' placement with the motive to minimize controller-controller and controller-switch latency while keeping the load balancing as the constraint. The CPP can be addressed by using NMR algorithm as follows.
In NMR algorithm, the solution of the optimization problem is determined by the

position of the NMRs. First, number of NMRs is initialized, with each NMR representing one of the combinations of k possible placement of controllers to be deployed. After initializing the NMRs, each of the controllers' positions is assigned to the switches in the network based on the shortest distance to the controllers, and a label for switches mapping is created for every controller. After mapping of the switches to the controller, the load of the controller is measured by using Equation 3.6. If the capacity of controller exceeds its limitation, the switch is assigned to the second nearest controller. As a switch is assigned to the possible nearest controller, the delay from a node to a nearest controller is measured in terms of distance. Then, the objective function Equation 3.7 is calculated using Equation 3.8, which represents the fitness of the NMRs. The best b breeders are selected based on the best fitness value and remaining NMRs are selected as workers. Also, the best position of the NMR is set as global best position. Then, until the termination criterion is met, the position of breeders and workers are updated. For all the breeders, the position is updated using Equation 3.3. Breeder NMRs are used for exploitation of search space; thus the search space of breeders is situated around those controllers' position which obtain the best fitness value. Hence, breeder NMR's position is always directed by the best NMR position. After the position is updated, the fitness of the breeders is updated using Equation 3.7, and if it improves the fitness, then the new position is remembered.

Similarly, for all the workers, the position is updated using Equation 3.2. The main purpose of worker NMRs is exploration of the search space. The exploration of the search space is based on the two random worker NMR positions; thus, the update of the position vector of worker NMRs is directed by the two randomly selected workers' positions. Thus, like a random search in the search space, worker NMR performs the exploration part. After updating the position of workers and breeders, their fitness is evaluated and only if the new fitness is better than the previous one, it is stored in the memory. Then, evaluation of the fitness of the NMRs is done and only the best NMRs are selected as the breeder for next iterations. The best among the breeders is memorized as the best solution for that iteration. Finally, when the termination criterion is met, the best breeder NMR presents the solution of the controller placement problem. Thus, the solution contains the position of the controllers, which essentially means the longitude and latitude of the controllers with switch to controllers mapping relationship and the best minimized value of the objective function.

Algorithm 1: Naked mole-rat controller placement problem

Input: $LatLong$, $nCont$, $SrtPathMtrx$ // Latitude & longitude, number of controllers, the shortest distance between each nodes

Output: $best\_latency$ (Optimized Cost), $FinalCtrlPos$, $MappingS2C$ // best value of minimized latency, latitude and longitude of the controllers, switch to controller mapping relationship

Initialization:

$nSwitch \leftarrow$ size of Latlong // total number of switches

$maxIter \leftarrow$ maximum number of iterations

$nPop \leftarrow$ NMR swarm size // total number of NMR population

$bp \leftarrow$ breeding probability // the breeding probability of the breeder NMR

$n_b \leftarrow nPop/5$ // population size of the breeder NMR

for i = *1 to* $nPop$ **do**

    $nmr(i).Pos \leftarrow$ randomly select nCont from nSwitch

    $label \leftarrow$ calculate mapping of switch to controller

    $fitness \leftarrow$ evaluate latency of $nmr(i)$

Optimization() // include function optimization

while *(iter <* $maxIter$*)* **do**

    /* sort NMRs according to fitness in ascending order */

    $X_{best} \leftarrow$ NMR with best minimum latency

    $L_{best} \leftarrow$ label of the best NMR // mapping of controller and switches

for i = *1 to* $nPop$ **do**

    if $U(0,1) > bp$ **then**

        update position using: $b_i^{t+1} = (1-\lambda)b_i^t + \lambda(X_{best} - b_i^t)$

    /* λ, a mating factor refers to random (0,1) */

    if *latency of* $b_i^{t+1} < b_i^t$ **then**

        update the new position to b_i^{t+1}

for $i = n_b$ + *1 to* $nPop$ **do**

    update the new position of: w_i^{t+1}

    if *latency of* $w_i^{t+1} < w_i^t$ **then**

        update the new position of: w_i^{t+1}

$iter \leftarrow iter + 1$

$FinalCtrlPos \leftarrow$ position of X_{best}

$MappingS2C \leftarrow$ label of L_{best}

$best\_latency \leftarrow$ fitness value of X_{best}

Algorithm 1 elucidates the flow of how NMRA works to solve the controller placement problem and the Algorithm 2 presents the steps of fitness evaluation. The optimization() function is called from Algorithm 1. This function provides the steps to calculate the best latency and optimal location. The positional update of the worker is based on L, which is the flight steps of the levy and the two randomly selected workers. NMRs while breeders are updated based on λ which is the random number between 0 and 1 and the best breeder, which is basically the best NMR. Here, each NMR represents the solution of the controller placement containing the coordinates of the nodes to be deployed.

Algorithm 2: Fitness evaluation

```
Input: ShrtPathMtrx, nmr.pos ; // shortest path matrix containing distance
  between the nodes, the original controller position
Output: fitness, label ;       // the average SC latency, mapping of switch to
  controller
for each nmr do
    for each node do
        find the shortest distance path to the controller;
        label = assign the node nearest to it
        Compute controller load capacity using Equation 3.6
        while controller load exceeds the capacity do
            assign switch to the next nearest controller
            update the label
    for each controller do
        find the shortest distance path to controller;
        evaluate the controller-controller latency using the path
    fitness = the combined global average latency using Equation 3.7
Return fitness, label
```

3.3 Traffic-Driven Controller-Load-Balancing over Multi-Controller Software-Defined Networking Environment

3.3.1 System scenario

The Ryu framework, which acts as the OpenFlow/SDN controller, is designed to run on a host computer connected over TCP to the simulated Mininet network topology. Python programming is used to implement the Ryu controller and the OpenFlow protocol 1.3. Table 3.2 provides a list of the particular testbed specifications employed.

Table 3.2: Testbed requirements

No.	Name	Specification
1	Operating System	Ubuntu 22.0.4 LTS (64 bits)
2	Ryu Controller (Ryu)	Version 4.34
3	Mininet Emulator (Mininet)	Version 2.3.0
4	OpenFlow Protocol [OpenFlow]	Version 1.3
5	Type of control plane	Hierarchical
6	Link Bandwidth	HS: 100 Mbps, SC: 1000 Mbps, SS: 10000 Mbps
7	Delays	PropD 2ms, processD 0.5ms, queuingD 1ms, controllerD 5ms
8	Queuing size	1000 packets (default)

The experimental network use case that is used in this study is shown in Figure 3.3. After setting up the basic network topology consisting of three clients (one sender and two receivers) in the mininet, the output is shown in Figure 3.4.

Three controllers, namely, one main (local) and two secondary local controllers, are connected to a global controller that monitors the CPU utilization of all controllers and switches roles if the threshold is exceeded. The figure 3.5 demonstrates three local controllers connected to a global controller.

3.3.2 Dataset generation

The distributed internet traffic generator (D-ITG) platform generates traffic flow data, which is then used to train ML models [158]. D-ITG is capable of producing IPv4 and IPv6 traffic by accurately modelling the workload of current internet

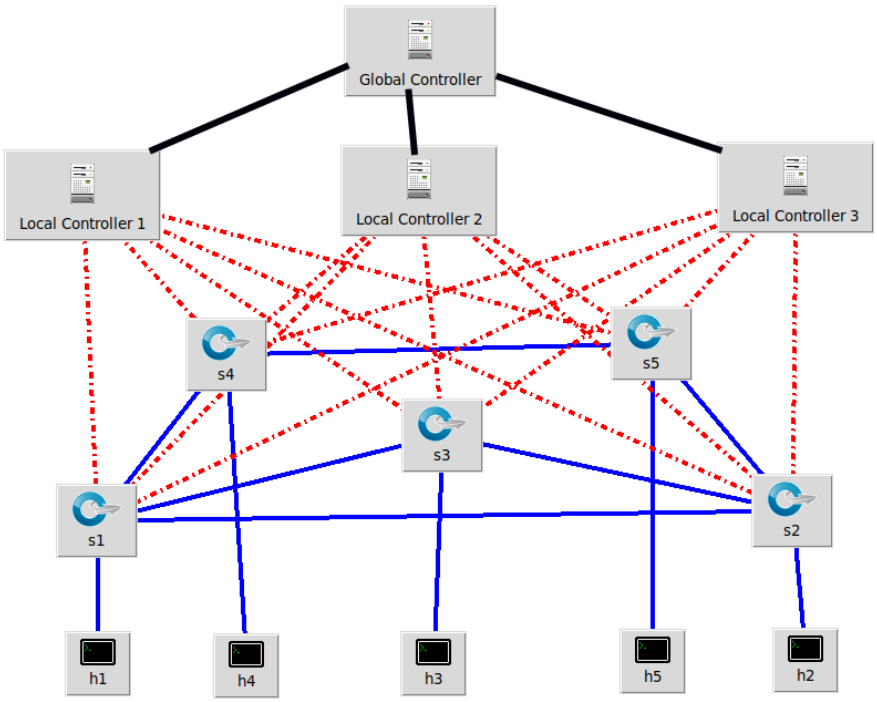


Figure 3.3: Experimental network use case

```
<Host h1: h1-eth0:10.0.0.1 pid=8090>
<Host h2: h2-eth0:10.0.0.2 pid=8092>
<Host h3: h3-eth0:10.0.0.3 pid=8094>
<Host h4: h4-eth0:10.0.0.4 pid=8096>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=8101>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=8104>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=8107>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None pid=8110>
<RemoteController c1: 127.0.0.1:6633 pid=8076>
<RemoteController c2: 127.0.0.1:6634 pid=8080>
<RemoteController c3: 127.0.0.1:6635 pid=8084>
```

Figure 3.4: Mininet output (dump)

```
INFO:__main__:Waiting for connection.....
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7f8aa9de2b00> address:('127.0.0.1', 55214)
line 208 msg is  {"cmd": "set_agent_id", "agent_id": 0}
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7f8aa9e0bc70> address:('127.0.0.1', 33416)
line 208 msg is  {"cmd": "set_agent_id", "agent_id": 1}
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7f8aa9e701c0> address:('127.0.0.1', 38266)
line 208 msg is  {"cmd": "set_agent_id", "agent_id": 2}
```

Figure 3.5: Output of the global controller

applications. The D-ITG program helps to mimic real-world traffic conditions by providing fine control over characteristics like packet size (PS), inter-departure time (IDT), and traffic allocation. Some limitations, such as multiport and burst traffic options, are not considered for this research, as this research is an attempt to check if multi-controller SDN improves QoS over single controller SDN using a traffic classification approach in a controlled environment. D-ITG can generate traffic at the network, transport, and application layers and supports both IPv4 and IPv6 traffic

generation.

The following types of traffic are used for the concept of traffic classification:

1. Telnet: It emulates the Telnet traffic of the real world with typical Telnet characteristics. It works with the TCP transport layer protocol.

2. DNS: It generates traffic with DNS characteristics simulating real-world scenarios and works with both UDP and TCP transport layer protocols.

3. Game: It generates counter-strike traffic with both active players and idle player scenarios. It is only compatible with the UDP transport Layer protocol.

4. Voice: It generates the traffic of voice over IP, emulating the characteristics of a voice call. It has various options to simulate different scenarios of a voice call.

5. Video: It generates the traffic of video by accessing a server playing Quake3 video.

Here, Ping, telnet, game, and DNS are all put in the "data" class to demonstrate the concept of traffic classification, while voice and video are put in the "voice" and "video" classes, respectively. These three types have been selected because almost all normal traffic generated by the D-ITG tool belongs to one of these classes.

A training dataset sample [159] is shown in Table 3.3. To save operating time, only the essential features are retrieved, while extraneous features are eliminated from the dataset because all of its contents are irrelevant for performing data pre-processing.

Table 3.3: Training dataset sample

Forward Packets	**Forward Average Packets/second**	**Reverse Packets**	**Traffic Type**
454	1.166666667	507	Voice
790	0.923076923	1018	Voice
856	0.105691057	508	Data
856	0.10483871	508	Data
856	0.09352518	507	Video
1622	0.320512821	507	Video

3.3.3 Proposed work flow model

Firstly, let us examine a scenario in which the model can be applied to enhance the overall QoS of the network. Consider a company that uses VoIP calls for customer service and grievance handling. In such a scenario, voice traffic from customers has the highest priority and needs to cross the network with minimum latency and maximum throughput. Other forms of traffic, such as data traffic and video traffic, from sources like Facebook and YouTube on their personal devices, should be given less priority without affecting the organization's performance. The proposed model may work well in this situation since it classifies traffic first and then simply balances the load according to the various types of traffic.

To achieve excellent performance and of QoS in modern SDN environments, efficient flow management of network traffic and load balancing across multiple controllers are essential. As network traffic continues to expand rapidly due to diverse applications and varying traffic patterns, real-time precise traffic management and load control across many controllers play a significant role in improving the QoS [160].

Additionally, to manage dynamic traffic patterns and guarantee optimal resource use, traditional load balancing solutions might not be adequate. In order to handle any controller failures or overload scenarios, load balancing must be conducted intelligently and adaptively in order to classify network traffic in real-time and perform load balancing based on the priorities and characteristics of the traffic [161].

Once the traffic has been classified, the packets are routed to the controllers that are responsible for the particular kind of traffic. The data is then sent to the destination address according to the priorities and queuing policies set up in the Ryu controller and OpenFlow switches.

The goal of this research work is to create a traffic-driven controller-load-balancing solution for multi-controller SDN systems. To achieve precise traffic classification and dynamic load distribution, ML techniques has been utilized.

This study has been broadly classified into two sections, namely traffic classification and load balancing.

3.3.4 Traffic classification model

For the purpose of traffic classification, various SL algorithms have been tested to find the novel model capable of performing multi-class traffic classification with

appreciable accuracy and precision while also keeping the inference time and resource utilization low. Some of the tested algorithms include logistic regression, SVM, K-NN, XGBoost, decision tree (CART), and OPTICS. The ML algorithms are selected based on the findings in [121, 123, 124] to find the best possible algorithm for the classification of data before the operation of load balancing. Since traditional SL produces acceptable results and advanced ML (for example, DL) requires more data, sophisticated hardware, and increased computational complexity, SL rather than DL is chosen here.

Some preprocessing are necessary to accurately classify the data. The preprocessing procedures include removing null values, separating feature sets and labels, converting traffic type categories from object types to numeric values through label encoding in the dataset for further manipulation, normalizing the feature sets to avoid overfitting, and selecting the best features from the feature sets that are available because the ML classification model is not able to analyze a large number of features in an efficient manner. Based on the minimun feature scores of several features, the features are selected. The feature scores of the top 8 features are utilized in this proposed system for traffic classification tasks.

In this system scenario, the main (local) controller performs traffic classification and load balancing while the secondary (local) controllers handle less critical traffic. Here, the proposed model itself collects training datasets using real-world scenarios.

The algorithm 3 starts with the collection of training datasets. For this, the Ryu controller is initiated in training mode and the topology is started. The dataset is created by using a simulation tool known as D-ITG that simulates different types of traffic in a network. The data are extracted as a CSV file, which is then fed to the preprocessing state, where necessary feature sets are extracted, where as unnecessary features are dropped for easy and efficient manipulation in subsequent states. After preprocessing, the dataset is fed to the traffic classification model, where the model is trained using various ML algorithms for real-time traffic classification in future implementations.

Algorithm 3: Proposed work flow model

```
Initiation of network in training mode
Generation of Training dataset
Data Pre-processing
Traffic Classification Model training
Send trained model to step 10
Get Real time traffic from external network
Process Network in real time classification mode
if Traffic already not classified then
    Send Traffic to Main controller for classification
if Main controller classify traffic then
    Update flow tables and send traffic to dedicated path based on flow tables
    Send traffic to switch connected to receiver host
else
    Send traffic directly through switches
Traffic received by receiver host
```

While evaluating the performance, in a binary case, since precision and recall do not consider the true negative elements, the binary F1 score uses the equations 3.11, 3.12, and 3.13 for each machine learning technique. The F1 score is the harmonic mean of precision and recall.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.10}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.11}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.12}$$

$$F1 - Score = 2 \times \frac{Precsion \times Recall}{Precision + Recall} \tag{3.13}$$

In multiclass cases, F1-Score should involve all the classes and requires a multi-class measure of Precision and Recall to be inserted into the harmonic mean. The

computation of precision, recall, and F1-score involves averaging across classes. The common averaging methods are Macro F1-Score, weighted-average F1-Score, and micro F1-Score [162].

To obtain the macro-averaged F1 Score in Equation 3.14, macro average precision and macro average recall are computed as the arithmetic mean of the metrics for single classes [163]. The macro-averaged F1-score is not necessarily between the macro-averaged precision and recall because the averaging is done after computing F1 per class.

$$F1 - Score_{macro} = \frac{1}{N} \sum_{i=1}^{N} 2 \times F1_i \tag{3.14}$$

Where, N is the total number of classes and F1i is the F1-score for class i calculated as:

$$F1 - Score_i = 2 \times \frac{Precsion_i \times Recall_i}{Precision_i + Recall_i} \tag{3.15}$$

Calculating the F1-score for each class and then taking the average weighted by the number of true instances (support) for each class, a weighted average F1-score is obtained by 3.16 and 3.17. This approach accounts for class imbalance by giving more weight to classes with more instances.

$$F1 - Score_{weighted} = \sum_{i=1}^{N} w_i \times F1_i \tag{3.16}$$

where,

$$w(i) = \frac{support_i}{\sum_{j-1}^{N} support_j} \tag{3.17}$$

is the weight on class i based on its support and F1i is as defined above. Like macro-averaging, the weighted F1-score is not guaranteed to lie between the weighted precision and recall due to the averaging process.

After training the model, it is ready to classify real-world traffic in real time. The "pickle" library is then used to save the learned model and perform real-time traffic classification. Once it has been suitably trained and evaluated on Google Colab, the Ryu controller is used to run the model in real-time classification mode. The mininet topology can be resumed after the controller has started running. Following that,

when traffic is sent between hosts, the controller classifies it according to its type and presents it with additional data. The traffic received in real time is classified into pre-defined categories, which are already labeled as "video, "voice," and "data," as shown in fig 3.6.

Flow ID	Src MAC	Dest MAC	Traffic Type
-3666110929039238057	00:00:00:00:00:03	00:00:00:00:00:01	data
2985619599229988943	00:00:00:00:00:03	00:00:00:00:00:01	data
6234584942743833136	00:00:00:00:00:03	00:00:00:00:00:01	data
8295897452636130751	00:00:00:00:00:01	00:00:00:00:00:02	video
1637700141731743840	00:00:00:00:00:02	00:00:00:00:00:01	data

Figure 3.6: Real time classification

Following completion of traffic classification by the main controller, the output of the classifier provided by the ML algorithm is used to produce the flow label of the traffic based on its type and is updated in the flow table of the OpenFlow switch. Based on the match provided by the OpenFlow switch, the groups are designed to carry out various specific actions on different types of received traffic. The OpenFLow switch manages the additional forwarding of traffic across various switches from source to destination if there is a match in the flow table. If there is no match, the Ryu controller is activated to extract the parameter and classify the traffic into the appropriate label that the controller has predefined during training. After being classified, a packet is then delivered to a switch to follow the routing procedure. The explanation of the controller–switch interaction includes a workflow diagram for better understanding is presented in Figure 3.7.

3.3.5 Traffic priority based load balancing

Despite the fact that there are many load balancing techniques in the state-of-the-art, a typical IP-based multi-class traffic classification is presented, and load balancing in a multi-controller SDN environment is discussed here.

Traffic priority-based load balancing is different from traditional load balancing schemes, where the decision of path selection is not made only on the basis of the shortest path or load on the controller or switch but also the severity of traffic is taken into consideration. The proposed load balancing algorithm 4 is shown below.

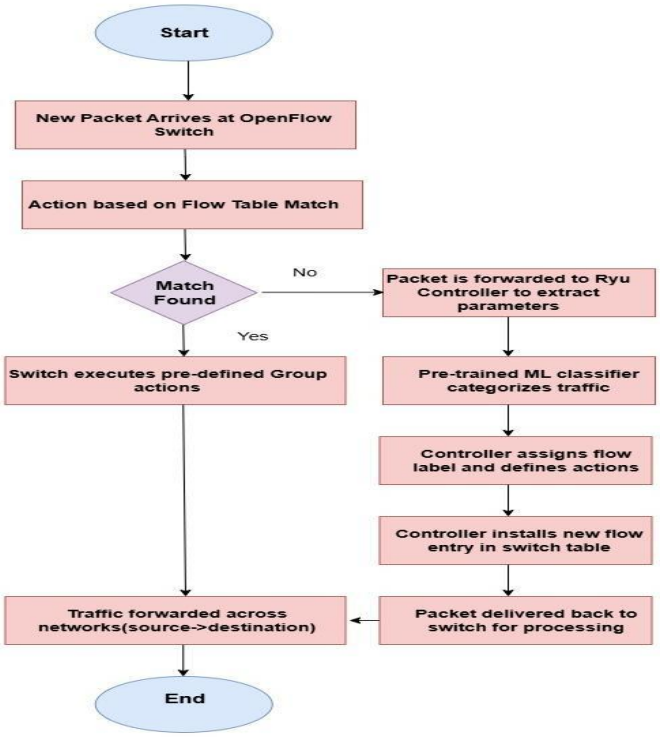


Figure 3.7: Switch Controller flow interaction WorkFlow diagram

Algorithm 4: Proposed load balancing algorithm in multi-controller SDN environment

```
Host start sending traffic to another host in network
Get incoming traffic from external network
OVS switches transfers unknown traffic packets from external network
while Traffic Level do
    if Video then
        Send traffic to dedicated path1 // send the video traffic
    if Voice then
        Send traffic to dedicated path2 // send the voice traffic
    if data then
        Send traffic to dedicated path3 // send the data traffic
Send traffic to the switch connected to the receiver host
Send traffic to the receiver host
Traffic received by receiver host
```

Each traffic type has its own path, and when congestion occurs, the traffic with the greatest priority is shifted to another path. Other types of traffic are either passed parallelly or blocked based on the path's capacity at the time.

The simple depth first search (DFS) algorithm, which analyzes the topology to find the path with the minimum cost between the source IP and destination IP, is used

to calculate the shortest path. Thus, using the pre-defined rules, the controller and path that satisfy the delay requirement for the specific type of traffic are then chosen.

Each switch's flow table and group table are updated to handle various types of traffic separately for the purpose of allocating different paths for different types of traffic. Then the intermediate switch sends the traffic to the terminal switch connected to the host, following the shortest path given by DFS from the host to the destination user.

The global controller to which both local controllers, main and secondary, are connected monitors the CPU utilisation of both controllers using the get_cpu_utilisations() function. CPU UTILISATION THRESHOLD = 70 is used to set the threshold at 70%. Generally, the common IT infrastructure resource utilization raises warnings after 70% utilization [164], [165]. Hence, the utilization of 70% is choosen; however, this is the simulation parameter value can easily be changed and set a different one for threshold.

The global controller runs on port 6633, which is the default port for Ryu-controller. The three local controllers run on ports 6634, 6635 and 6636, respectively. Using self.global_port, the local controller connects to the global controller by listening on its port. If the utilisation of the main controller exceeds the defined threshold value, i.e., 70%, the perform load balancing() function shares the classification and load balancing roles of the main controller with the secondary controller. The function send_role_request is called in order to accomplish this.

Further, the global controller continues monitoring the CPU utilization of both controllers, and if the threshold on the main controller drops below 70%, the complete role of traffic classification and load balancing is redirected to the secondary controller for handling upcoming packets. The threshold for this research is chosen at random and can be adjusted by the network administrator based on the network's convenience or performance requirements. Furthermore, the performance of the proposed approach may be verified using various threshold values. The detailed flow of controller switching action in the case of CPU utilization exceeding the threshold is shown in Algorithm 5.

Algorithm 5: Proposed controller switching during overload

```
Main controller operating in real-time mode and secondary controllers are active
while Send the CPU utilization status of all three controllers to the global controller do
    if CPU utilization of main controller is greater than 70% then
        Send upcoming traffic to secondary controller Update flow tables of switch based on load balancing rules
    else
        Continue handling traffic through the main controller
    Measure CPU utilization of both main and secondary controller
```

3.4 Attack Detection and Mitigation over an Optimal Multi-Controller Placement SDN Environment

3.4.1 Algorithms used for optimal controller placement

Determining how many controllers to have and where to place them in the network for optimum performance is one of the primary issues in SDN [24]. K-means++ [166] and Ordering Points to Identify the Clustering Structure (OPTICS) [167] are implemented to achieve optimal controller placement.

K-means++ is an improved version of K-means algorithm, despite both solving a clustering problem, K-means++ offers an additional edge by making informed initialization of initial centroids (makes use of squared distance metric for centroid selection), yielding better clustering results. K-means places k centroids (one for each cluster) randomly, so there exists a probability of bad initialization, leading to a bad clustering result. K-means++ overcomes this initialization problem [24]. For K-means++, number of clusters (k) and number of data points are provided as input, and for the output, it returns the optimal number of clusters for the given data points. Elbow-method is used for the identification of optimal number of clusters here, details of its implementation will be discussed in section 4.3.1.

Similarly, OPTICS is an improved version of the density-based spatial clustering of applications with noise (DBSCAN) algorithm [168] used to obtain density based clusters [169] in spatial data. Density based cluster means clustering data points on

the basis of region of high density or low density. Clusters can be arbitrarily shaped in data space [170]. Ankerst et al. [167] put forth an OPTICS-based cluster analysis technique. Unlike K-means++, for input field, it does not require initial number of clusters, instead it needs number of data points and two hyper-parameters, namely, eps = maximum set neighborhood distance, and $minpts$ =minimum number of points to be classified as a dense region, to be provided.

The process of determining opt imal controller placement is shown in Figure 3.8.

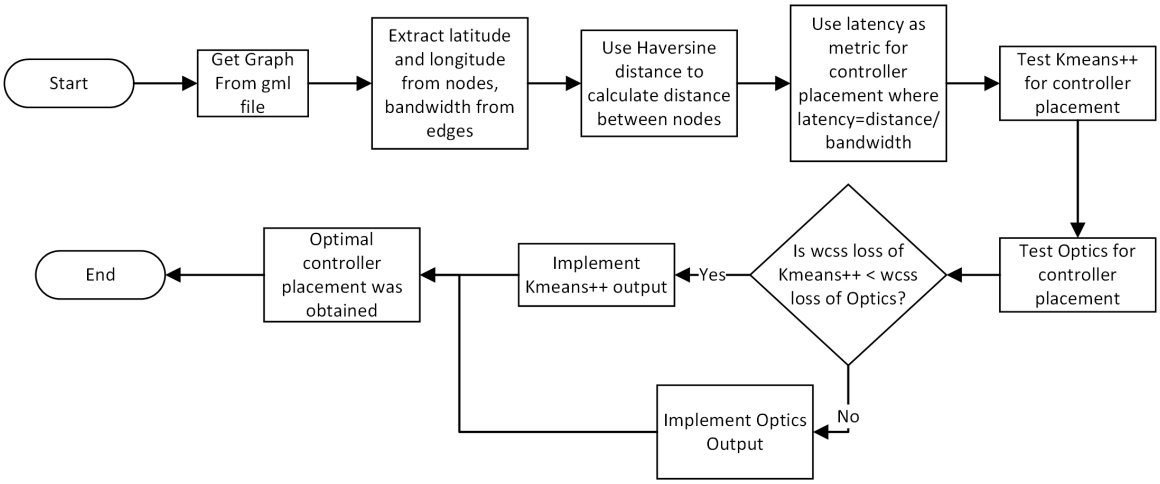


Figure 3.8: Flowchart for optimal controller placement implementation

Algorithm 6 shows the implementation of K-means++ and OPTICS to obtain an optimal controller placement.

$$\text{dH(lat1,lat2,long1,long2)} = 2(r)arcsin\sqrt{sin^2(\tfrac{lat_2-log_1}{2}) + cos(lat_1)cos(lat_2)sin^2(\tfrac{long_2-long_1}{2})} \quad (3.18)$$

$$d(x_1, x_2) = \frac{Haversine\ distance\ between\ x_1\ and\ x_2\ (in\ Km)}{bandwidth\ between\ x_1\ and\ x_2\ (in\ Mbps)} \quad (3.19)$$

where,
$Haversine\ distance(lat1, lat2, long1, long2) = dH(lat1, lat2, long1, long2)$,
$radius$ =6371 km,
$lat2$ =latitude of the node 2, $long2$ =longitude of the node 2,
$lat1$ =latitude of the node 1, $long1$ =longitude of the node 1, and

Algorithm 6: Controller placement algorithm

Input: $GraphG(V, E)$

```
/* Graph without controllers, where G=Graph, V=Set of vertices and E=Set of
   Edges                                                                    */
/* GetNeighbours(v, Graph(vertex,edges)) returns all the vertices connected
   to v                                                                     */
/* L = Minimum Within-Cluster Sum of Squares (WCSS) loss value              */
```

$L1 \leftarrow 0,\ L2 \leftarrow 0$ /* L1=WCSS loss using OPTICS, L2=WCSS loss using K-means++ */

$G1(V,E),\ C1(V) \leftarrow OPTICS(G(V,E), distance)$ /* G1(V,E)=Graph with controller placement using OPTICS,C1(V)=list of controller vertices from OPTICS */

$G2(V,E),\ C2(V) \leftarrow Kmeans++(G(V,E), distance)$ /* G2(V,E)=Graph with controller placement using K-means++,C2(V)=list of controller vertices from K-means++ */

for *each* $v \in C1(V)$ **do**
    U ← GetNeighbours(v,G1(V,E))
    for *each* $u \in U$ **do**
        $L1 \leftarrow L1 + distance(u, v)$

for *each* $v \in C2(V)$ **do**
    U ← GetNeighbours(v,G2(V,E))
    for *each* $u \in U$ **do**
        $L2 \leftarrow L2 + distance(u, v)$

if $L1 < L2$ **then**
    G(V,E) ← G1(V,E)
    $L \leftarrow L1$

else
    G(V,E) ← G2(V,E)
    $L \leftarrow L2$

return $L, G(V, E)$

$distance(x_1, x_2) = d(x_1, x_2)$, where $x1$ and $x2$ are nodes present in the selected Graph (say node 1 and node 2)

The haversine distance formula from Equation (3.18) calculates the shortest distance between two points on a sphere. In Equation (3.19), latency is a function of *distance*, defined as how fast packets traverse between two nodes, and is directly proportional to the physical distance separating these nodes. However, if the bandwidth between the nodes is increased, data transmission occurs at higher rates, resulting in reduced delay between consecutive packets, indicative of an inverse relationship.

The workflow diagram of the designed system shown in Figure 3.9 summarizes the

entire workflow of network traffic (both benign and attack), providing a visual representation of how the multi-controller setup is organized to detect and mitigate DDoS attacks efficiently. Communication scheme between different components of SDN environment, Host-Switch, Switch-Controller, and Controller-IDS is shown. A switch forwards the packets from a host to the respective interface using Flow table. Switches also advertise the flow information to their associated controller in every 10 seconds. Besides controlling the switch's flow table, a controller also extracts features from the flow information that is then transmitted to the central IDS. IDS performs the task of classification of the traffic, benign/attack, and in case of an attack traffic, it broadcasts the attacker source information to each of the controllers. Upon receiving this attacker information, the controller activates the mitigation module and requests the installation of a flow in the respective switch to block the port of the attacker. Switch upon receiving this request will install this flow, and as a result, the attacker will be blocked for a predefined duration of 10 minutes. For normal traffic, no specific action is performed, and the traffic follows a normal natural route. A more detailed explanation of the detection and mitigation mechanism is given in Section 3.4.9.

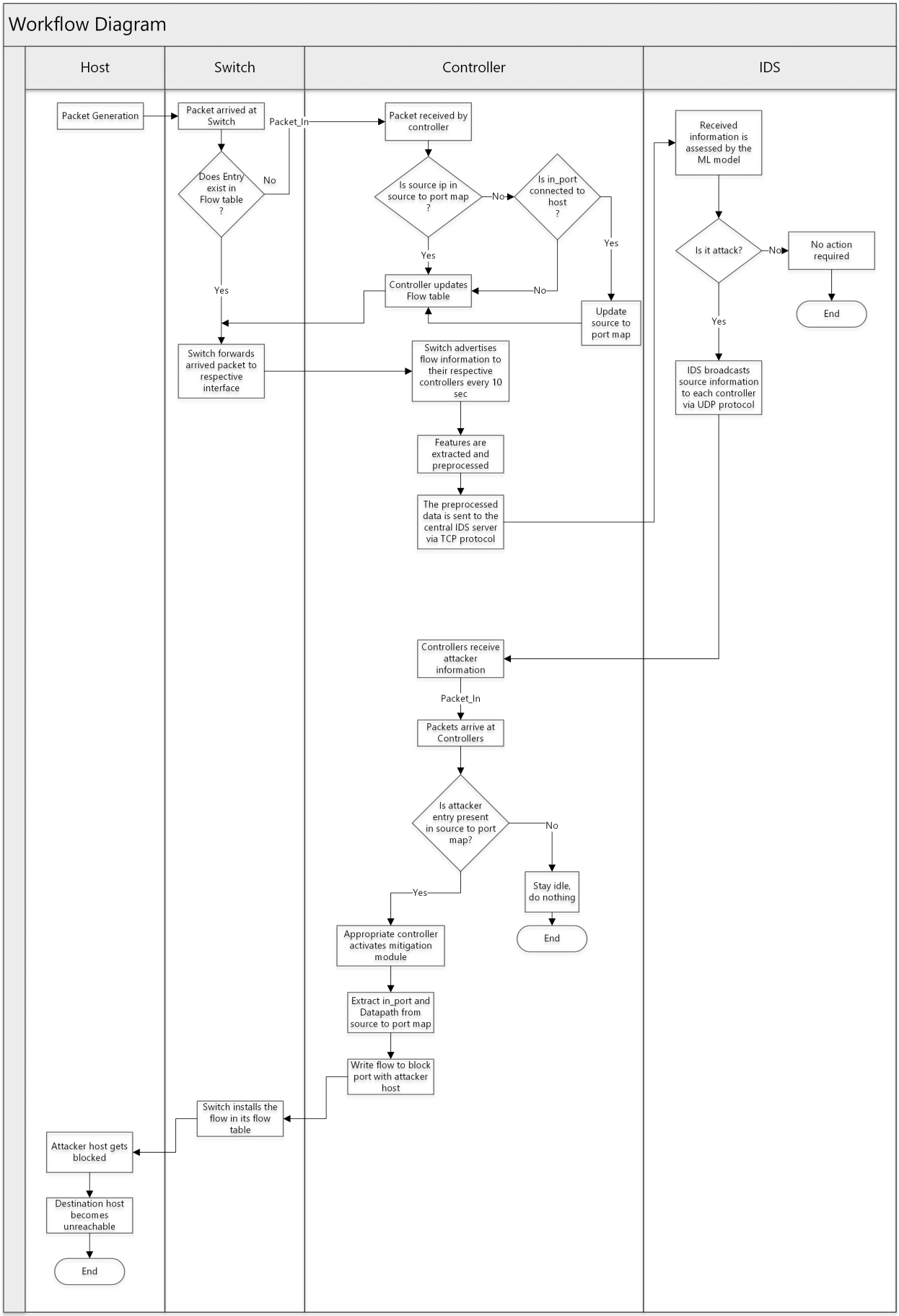


Figure 3.9: Workflow diagram of the designed multi-controller based system

3.4.2 Development of network topology

GML files for the Aarnet and Savvis topologies are downloaded from the Internet Topology Zoo [171]. K-means++ and OPTICS algorithms are used to determine the optimal number and positions of the controllers. Controllers are placed at centroids of each cluster such that a controller controls switches (nodes) falling in that cluster. An outbound control mechanism is implemented, which means that data and control packets travel through different channels in the network. Following this, each node is assigned as an OpenFlow switch with a designated number, aligning them with the controllers determined by the clustering results of the best between K-means++ and OPTICS. This process results in an optimized SDN setup with strategically placed controllers and neatly assigned switches based on the clustering algorithm. Subsequently, two hosts are assigned to each OpenFlow switch and numbered as $(2 * switch\_number - 1)$ and $(2 * switch\_number)$.

3.4.3 Generation of dataset

The DDoS attack SDN dataset [172], designed specifically for SDN, is studied, and insights for relevant features are obtained. Also, the importance of ICMP type is realized by studying experiment [173].

The experiment is carried out on a Dell laptop (i5, 16GB RAM, Ubuntu 22.04.02 LTS 64bit). Mininet (version 2.3.0.dev6) is used to emulate a SDN environment. For emulation, IDS system, controller, switches, hosts are turned active. The switch used here is an OpenFlow switch capable of supporting Openflow Protocol. The setup for the dataset generation can be referred from Figure 3.10.

With these features in consideration, Savvis topology is used to generate Benign and DDoS traffic-containing datasets. ICMP (Pings) and custom sockets for TCP/UDP are used for the Benign dataset. For the DDoS dataset, hping3 floods (ICMP, TCP, and UDP) are used.

The malignant dataset is appended to the benign dataset and sorted according to the time attribute. The merged dataset contains both types of data. To distinguish between attack and normal data, labels 1 for attack data and 0 for benign data are assigned.

3.4.4 Details of the dataset generated

For Benign Dataset, it takes 110 minutes to generate the dataset. It contains 100,718 rows and 19 columns. While for Malignant Dataset, it takes 8.3 minutes to generate the dataset. It contains 33,613 rows and 19 columns. After merging the both, a total of 436,331 data samples are obtained with 19 features and size 53.69 MB. The attributes of this dataset is shown in Table 3.4. Column 0 to 17 represent independent features, and column 18 is the dependent feature.

Table 3.4: Details of generated dataset

Column Number	**Feature Name**	**Description**
0	*time*	time of arrival at the controller
1	*switch_id*	datapath id of switch
2	*ip_src*	source IP address
3	*src_port*	TCP or UDP port number
4	*ip_dst*	Destination IP address
5	*dst_port*	TCP or UDP port number
6	*ip_proto*	IP protocol
7	*icmp_type*	type of ICMP packet
8	*flow_duration_sec*	active time period of flow in sec
9	*flow_duration_nsec*	active time period of flow in nsec
10	*remaining_idle_time*	Remaining idle time of flow
11	*remaining_time*	Total remaining time of flow
12	*packet_count*	Total packets
13	*byte_count*	Total bytes
14	*packet_count_per_second*	Number of packets in 1sec
15	*packet_count_per_nsecond*	Number of packets in 1nsec
16	*byte_count_per_second*	Number of bytes in 1sec
17	*byte_count_per_nsecond*	Number of bytes in 1nsec
18	*label*	benign(0) or attack(1)

To the best of my knowledge, this dataset is the first collected in a multi-controller environment and does not require any packet capturing or monitoring overhead, which would add unnecessary strain on the controller. Instead, this dataset relies solely on reading the statistics from flow tables provided by OpenFlow switches. This approach helps reduce congestion in the control plane, unlike those that utilize CIC datasets.

3.4.5 Dataset preprocessing

The preprocessing involves eliminating rows containing any of $nan, inf, -inf$ values, reducing dataset size by changing data types wherever possible for faster processing, sorting the entire dataset using the *time* feature, performing feature selection, and dropping irrelevant columns numbered 1, 2, 3, 4, and 5 using correlation analysis to obtain necessary insights, performing a train (80%) and test (20%) split, resampling the training set to balance binary class distribution, and normalizing for optimized training and lower loss.

The processed dataset is then used for training ML models and also for testing the performance. Finally, the best-performing ML model is used for the IDS system.

3.4.6 Machine learning algorithms used for training dataset

ML, a subfield of AI, focuses on the development and use of statistical algorithms to generalize data and perform tasks without explicit programming. ML algorithms have been widely applied to various classification and prediction problems. In this study, several classification algorithms are evaluated, including Naive Bayes, XGBoost, KNN, RF, NN, and LSTM.

3.4.7 Data and control plane communication

The obtained topologies comprises multiple paths between hosts. To address this problem, the topology is transformed into a minimum spanning tree based on the distance (x1,x2) metric as shown in equation (3.19) to eliminate redundant paths.

Ryu controllers are programmed to map IP address and port number in OpenFlow switches. The mapping enables the switches to relay messages between terminals. Further, the switches replies with flow table statistics to their respective controllers every 10 seconds. In this case, controllers only process the forward-flow (Tx) advertisements to pass on to IDS. Controllers ignore the flows created in response to requests like ICMP and TCP. This ensures that flows in only one direction are processed and prevents false alarms due to burst reply packets.

3.4.8 Controller-IDS integration

Using Mininet, an optimal number of controllers are strategically placed, each assigned a consecutive IP address starting from 127.0.0.2. Additionally, a dedicated

host is emulated for testing IDS functionality, with IDS being placed at the centre of all controllers.

To facilitate communication and transfer of CSV data between the controller and IDS, a TCP server on port 9998 is initiated on the IDS. Once the file transfer is completed, the connection is terminated. Subsequently, a new TCP server on port 9999 is established on the IDS to receive Flow-Statistics data intended for AI models.

IDS, upon predicting if the received data is either an attack or normal, encapsulates information containing the attacker's IP address into a UDP datagram. This datagram is then sequentially sent to each controller at port 8888 if the prediction indicated an attack. If the prediction is normal, no information is sent to the controllers.

Manso et al. [174] propose an IDS that automatically identifies multiple DDoS attacks and alerts a SDN controller when an attack is identified. Reducing the quantity of control events that are sent to the SDN controller from the IDS module is crucial. If not, the SDN controller gets overloaded and takes too long to detect and mitigate a DDoS attack.

Congestion represents a critical factor in determining the efficacy of an IDS. The persistent transmission of messages from switches to the controller or packet capturing devices are well known to cause congestion in the control plane. To mitigate this issue, a multi-controller environment with controllers positioned strategically. In this setup, data flows from switches are allocated to their respective controllers. Thus, no single controller is burdened by a substantial volume of advertisements from the data plane, regardless of whether during an attack or under normal conditions.

Algorithm 7: IDS detection algorithm

Input: Flow Packet, C_n threshold = 5

```
/* C_n=List of controller IP Addresses                                    */
/* Extract_features() extracts relevant features flow statistics packet   */
/* XGB.Predict() classifies Benign/Attack using XGBoost                   */
/* GetMax_count() returns source IP of entry having maximum count and
   exceeding minimum threshold                                            */
victim[] ← NULL
while IDS is active do
    Receive Flow statistics from controllers every 10 seconds
    Frame ← Extract_features()
    for data ∈ Frame do
        prediction ← XGB.Predict(data)
        if prediction = 1 then
            victim[data.dst_ip] ← (data.src_ip,count+1)
        Attack_source,Attack_destination←GetMax_count(victim)
        victim[Attack_destination].count ← 0
        IDS.sendall(C_n,Attack_source)
        Reset victim[ ] every 150 seconds
```

In Algorithm 7,

C_n =List of controller IP Addresses

$Extract\_features()$ extracts relevant features flow statistics packet

$XGB.Predict()$ classifies Benign/Attack using XGBoost

$GetMax\_count()$ returns source IP of entry having maximum count and exceeding minimum threshold

Algorithm 7 describes the IDS detection process during a DDoS attack. An empty victim list is initialized in IDS. The IDS continuously monitors the flow statistics received from the controllers, updated every 10 seconds. The flow information originated from the Open flow switches and it makes up to the IDS via the respective controllers. The IDS extract relevant features from the frame for analysis. Using XGBoost, each flow is classified as either a Benign traffic (label 0) or an attack

traffic (label 1). If an attack is detected, attack count for the victim (using destination IP) is increased. The function GetMax_count() finds the source IP that has attacked the victim the maximum number of times. At this point, the attacker IP to be blocked is well determined. Now, the victim count is reset and IDS broadcasts the source IP to all the controllers. After every 150 seconds, the victim list is reset to free up memory and to prevent false positive for any legitimate traffic. However, once the IDS broadcasts the source IP flagging it as an attacker, it will remain blocked for 10 min.

An IP is flagged as malicious only after surpassing a threshold of minimum detection (set to 5, arbitrary assumption), reducing the likelihood of blocking legitimate traffic. If the threshold is set high the attack is persistent, increasing false negatives. Conversely, if the threshold is low, false positives increase. This parameter can be adjusted according to the use case and type of network. Additionally, the system resets counters periodically to ensure efficient memory usage and restoring blocked devices.

3.4.9 DDoS attack detection and mitigation in emulated SDN environment

DDoS attacks are common against SDN due to its complex and central architecture, especially at the control layer, where they can affect the entire network by using numerous resources at specific times [175].

A custom DDoS Python script is launched using the Scapy library (v.2.4.4) from one of the hosts. This script differs from the one applied in dataset generation because the attack rate (up to 30 packets per second) varies in comparison to the hping3 attack rate (up to 56 packets per millisecond) utilized for generating the dataset. Varying the rate is done to assess the generalizing capability of the trained model.

Figure 3.10 illustrates attack generation, detection and mitigation process in a multi-controller environment. The segregation of three planes in SDN, control, data and application plane is shown. Host and openflow switches lie in data plane; controllers lie in control plane and the central IDS lies in application plane. Cluster area represents the domain of a controller. Communication between switches and controller is via by Southbound APIs and between controllers and IDS is via Northbound APIs.

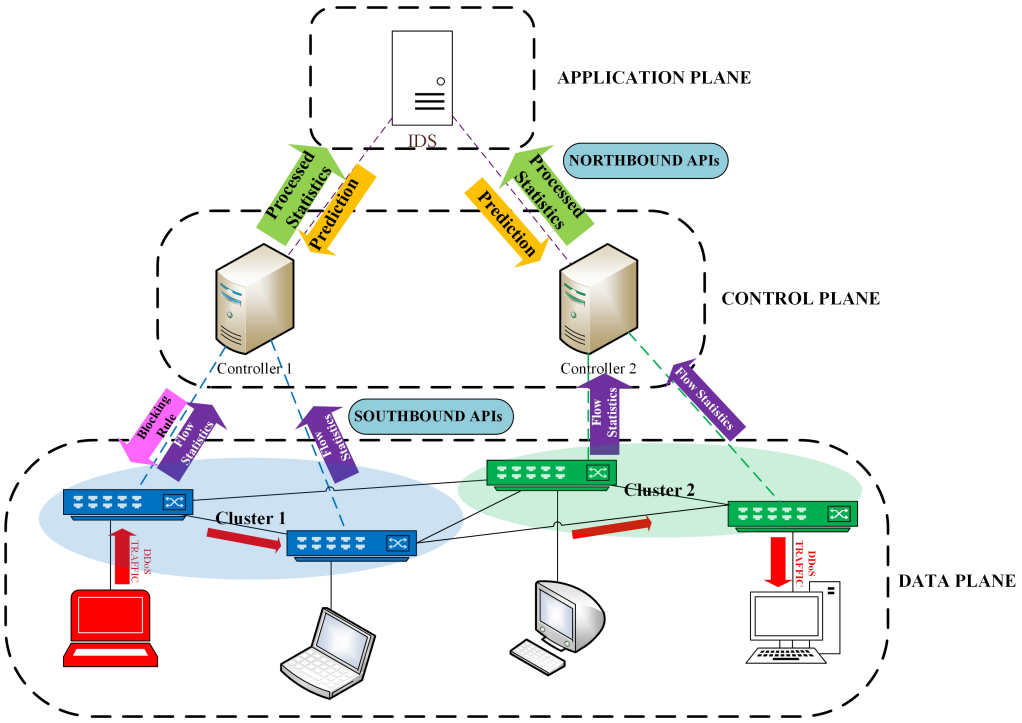


Figure 3.10: System diagram of multi-controller SDN setup.

When a host generates a DDoS traffic intended to another (victim) host located in different cluster, firstly, the packet goes through the associated switch. Switch then forwards flow data/statistics to its respective controller (here controller 1). Controllers process the data/statistics by transforming numerical values into floating point and encapsulating them for further transmission. It passes processed statistics to the central IDS. IDS uses this processed information in XGBoost algorithm and makes a prediction. If the prediction classifies the traffic as "attack" (prediction value of 1), this prediction is relayed to all the controllers. The controller with the attacking host in its domain installs a blocking rule in the switch, eventually the attacker gets blocked.

To ensure optimal performance and prevent memory exhaustion, the dictionary is cleared at regular intervals, specifically, every 10 minutes. This periodic clearing of the dictionary helps maintain system efficiency and prevents degradation of network performance.

The entire process of detection and performing mitigation it takes minimum 0 sec and maximum of 10 seconds. A model was trained using dataset from Savvis topology, and this trained model was used for detection and mitigation of DDoS attacks in both Savvis and Aarnet networks. Algorithm 8 shows how a controller blocks an attacker.

Algorithm 8: Attack mitigation algorithm

Input: Attacker IP Address, C_n

```
1  Function Alert_Controller:
2      for C_i ∈ C_n do
3          Source_ip ← C_i.receive(IDS)
4          C_i.store(source_ip)
5      return None
6  Function Mitigation_Module:
7      port_map[] ← NULL
8      Packet_In_Event
9      (datapath, switch_id, src_ip, in_port) ← Extract_features()
10     if in_port is connected to a host then
11         port_map[source_ip] ← (datapath, in_port)
12     attacker ← controller.load()
13     if attacker in port_map then
14         controller.block(port_map[attacker].datapath ,
             port_map[attacker].in_port)
15         controller.erase()
16     Reset port_map every 10 minutes
17     return None
```

In Algorithm 8,

- C_n=List of controllers

- C_i.store() stores the received IP into memory

- controller.load() loads stored IP from the memory

- controller.erase() clears the memory

- controller.block() creates flow entry to block the DDoS traffic

If any key in the port_map matches with *attacker*, it means that the attacking host lies in controller's domain, and if there is no match then the controller will remain idle since the attacking host is not in their domain. For the controller containing the attacking host, a blocking rule to block DDoS traffic is created by a flow entry using controller.block() module. Only the associated switch (identified

from the datapath) installs this blocking flow in its flow table. As soon as this flow is installed, the attacking host gets blocked. Meanwhile, the controller deletes old attack records from its memory using controller.erase() ensuring its memory is not overloaded. After a period of 10 minutes (an arbitrary assumption), the port_map dictionary is cleared. This prevents some genuine traffic from being blocked indefinitely. However, if the attacker still continues to send DDoS traffic, it will again be flagged and blocked by the same mechanism.

3.5 Chapter Summary

This chapter outlined the NMRA-based MCPO and traffic priority-based load balancing in different real-world topologies. A joint study of optimal controller placement system design and DDoS detection and mitigation in a multicontroller SDN environment is briefly discussed. Similarly, the development of mininet-eumulated network topology, dataset preparation and prepprocessing, necessary assumptions for simulations and analysis, tools and techniques, the development of system scenarios, proposed algorithms, and workflow models used in the whole research have been presented with the overall steps of the research work.

Chapter 4

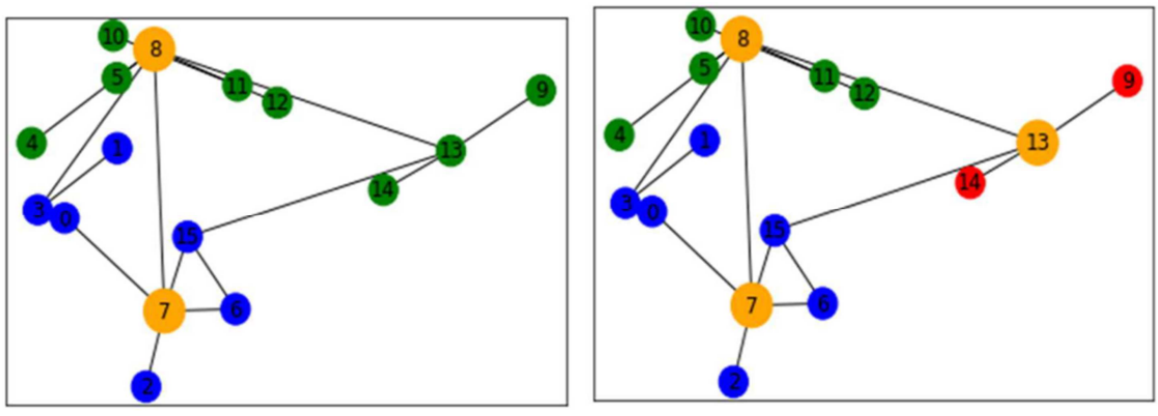
Results and Analysis

Conceptual theoretical formulations and research methodology based on the defined objectives were presented in Chapter 3. The outcome of experimental outputs and empirical/numerical/emulation analysis with findings and contributions are specially discussed in three different sections in this chapter.

4.1 Multicontroller Placement Optimization

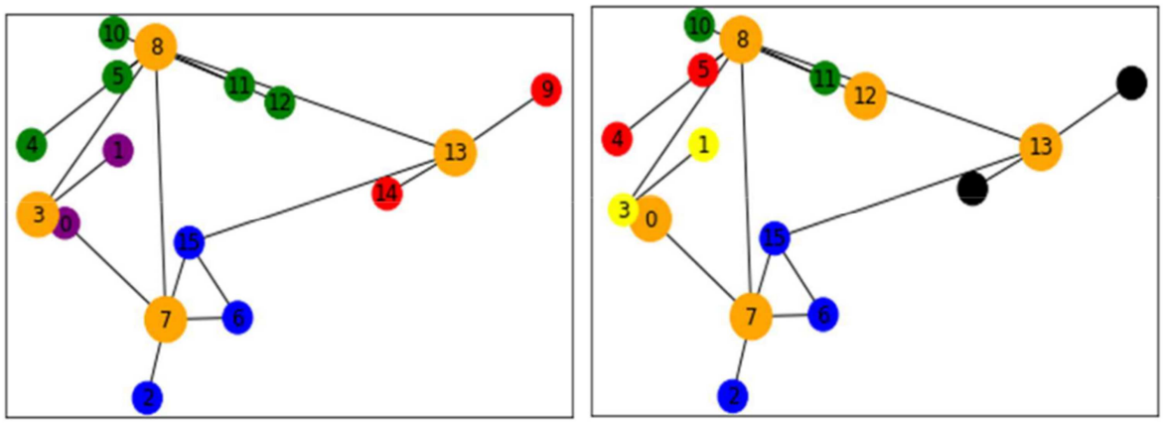
4.1.1 Case I. SC latency in Ernet topology

In this case, both the NMR and Bat algorithm perform better in terms of the exact same placement for the controllers. The number of controlled switches at different number of nodes for different number of controllers from Figure 4.1 and Figure 4.2, are shown in Table 4.1.



(a) (b)

Figure 4.1: Controller placement for (a) k=2 SC latency (b) k=3 SC latency



(a) (b)

Figure 4.2: Controller placement for (a) k=4 SC latency (b) k=5 SC latency

Table 4.1: Controller and switches placement details of Ernet topology

Number of controllers (K)	Number of controlled switches				
	Node(8)	Node(7)	Node(13)	Node(3)	Node(12)
1	Master Node	-	-	-	-
2	9 switches, Nodes: 4,5,8-14	7 switches, Nodes: 0-3,6,7,15	-	-	-
3	6 switches, Nodes: 4,5,8,10,11,12	7 switches, Nodes: 0-3,6,7,15	3 switches, Nodes: 9,13,14	-	-
4	6 switches, Nodes: 4,5,8,10,11,12	4 switches, Nodes: 2,6,7,15	3 switches, Nodes: 9,13,14	3 switches, Nodes: 0,1,3	-
5	3 switches, Nodes: 4,5,8	4 switches, Nodes: 2,6,7,15	3 switches, Nodes: 9,13,14	3 switches, Nodes: 0,1,3	3 switches, Nodes: 10,11,12

The Figure 4.3 illustrates the upshots on average SC latency with the increase in the number of the controllers. This can also be represented in Table 4.2. Thus, it can be concluded that, with the increase in number of the controller's, the average SC latency keeps on decreasing until there is a controller for a switch. Compared to a single controller, the greatest significant improvement in SC latency occurs when there are two controllers, resulting in a nearly 40% reduction in latency. The further considerable decrease in SC latency is seen until k=4, as each controller resides in the node's position with the maximum degree available.

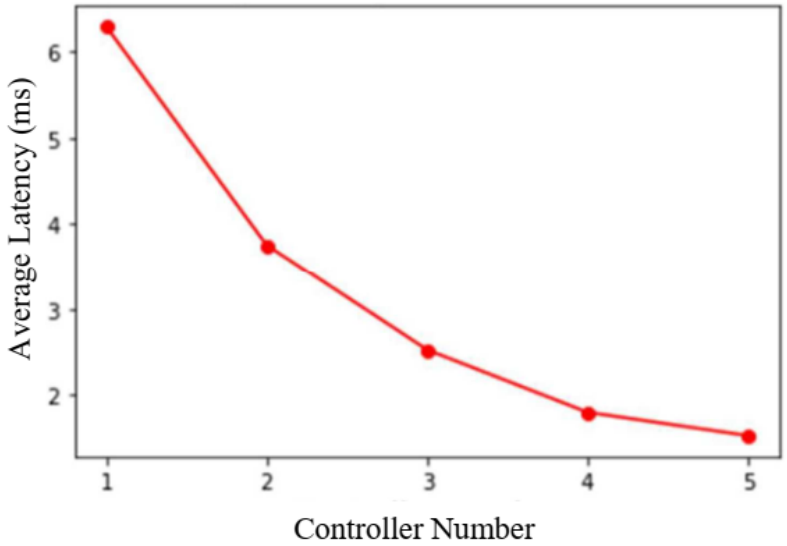


Figure 4.3: Average SC latency with increasing number of controllers for Ernet topology.

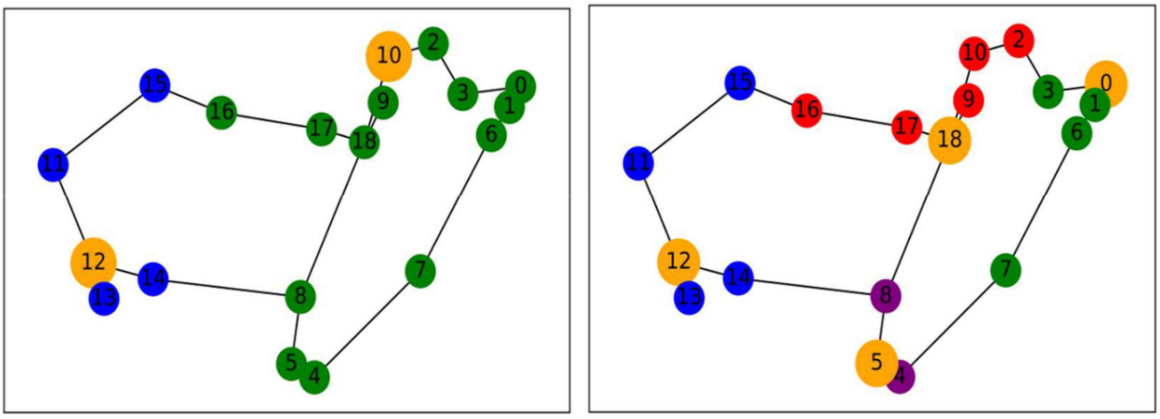
Table 4.2: Average SC latency with increasing number of controllers for Ernet topology.

Controller count (k)	**Average SC latency (ms)**	**% decrease in latency from previous**
1	6.28	
2	3.75	40.3
3	2.52	32.8
4	1.8	28.6
5	1.53	15.0

4.1.2 Case II: SC latency in Savvis topology

When considering SC latency only, the placement of the controllers is same for k=1, k=2, k=3, and k=4, thus both the algorithms have achieved the same SC latency. Figure 4.4 and Table 4.3 show the Savvis topology controller placement considering SC latency for both NMR and BAT in case of k = 2 and k = 4.

The Figure 4.5 portrays the SC latency decreasing with increasing number of the controllers. This behavior is clearly mentioned in Table 4.4. Here, the NMR algorithm slightly performs better in average SC latency when compared to the BAT algorithm.



(a) (b)

Figure 4.4: Controller placement for (a) k=2 SC latency (b) k=4 SC latency

Table 4.3: Controller and switches placement details of Savvis topology

Number of controllers (K)	Number of controlled switches				
	Node(10)	**Node(12)**	**Node(18)**	**Node(0)**	**Node(5)**
2 (BAT/NMR algorithm)	14 switches, Nodes: 0-10,16-18	5 switches, Nodes: 11-15	-	-	-
4 (BAT/NMR algorithm)	-	5 switches, Nodes: 11-15	6 switches, Nodes: 2,9,10,16-18	5 switches, Nodes: 0,1,3,6,7	3 switches, Nodes: 4,5,8

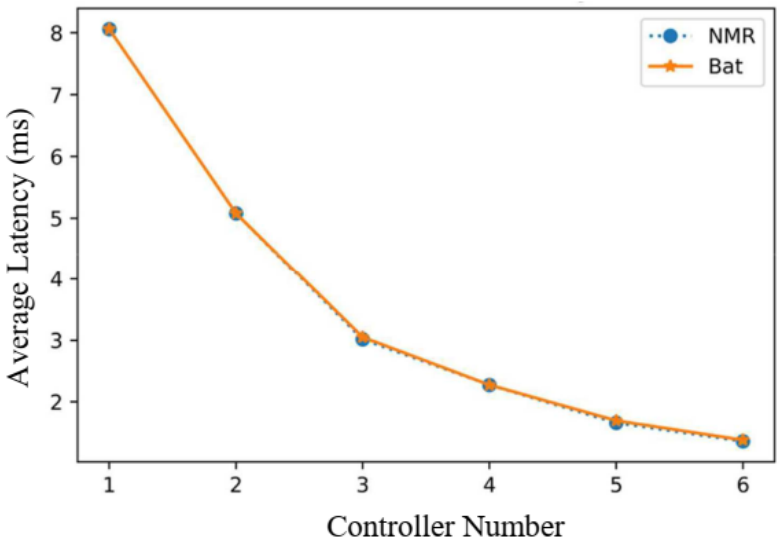


Figure 4.5: Average SC latency with increasing number of controllers for Savvis topology.

Table 4.4: Average SC latency with increasing number of controllers for Savvis topology.

Number of controllers (k)	Average SC latency (ms) NMR algorithm	Average SC latency (ms) BAT algorithm	% decrease in latency from previous
1	8.06	8.06	
2	5.08	5.08	37.0
3	3.01	3.05	40.8
4	2.27	2.27	24.6
5	1.66	1.69	26.9
6	1.38	1.38	16.9

4.1.3 Average SC and CC latency vs. number of controllers

If the consideration is given to combined SC latency and CC latency, there are two scenarios: (i) 0.9 SC and 0.1 CC (ii) 0.8 SC and 0.2 CC. The analysis is topology specific and mentioned below:

I. For Ernet topology with 0.9 SC and 0.1 CC:

In this scenario, 90% priority is given to average SC latency and 10% to average CC latency. For two controllers, i.e. k=2, and three controllers, i.e. k=3, also mentioned in Table 4.5, both NMR and BAT algorithm has the exact same placement and thus the same latency. However, due to addition of the CC latency, the algorithms focuses on the CC latency as well.

Table 4.5: Ernet: controller placement for K=2 and 3 using both NMR and BAT

Number of controllers (K) at 0.9SC + 0.1CC	Number of controlled switches			
	Node(8)	Node(0)	Node(3)	Node(7)
2 (NMR/BAT) algorithm	9 switches, Nodes: 4,5,8-14	7 switches, Nodes: 0-3,6,7,15 switches	-	-
3 (NMR/BAT) algorithm	9 switches, Nodes: 4,5,8-14	-	3 switches, Nodes: 0,1,3	4 switches, Nodes: 2,6,7,15

Also, the illustration of both NMR and BAT algorithm as described in Table 4.6 for the placement of four controllers shows the latency of NMR is better than BAT algorithm.

Table 4.6: Ernet: controller placement for K=4 using both NMR and BAT

Number of controllers (K) at 0.9SC + 0.1CC	**Number of controlled switches**				
	Node(8)	**Node(3)**	**Node(13)**	**Node(7)**	**Node(6)**
4 (NMR algorithm)	6 switches, Nodes: 4,5,8,10-12	3 switches, Nodes: 0,1,3	3 switches, Nodes: 9,13,14	-	4 switches, Nodes: 2,6,7,15
4 (BAT algorithm)	6 switches, Nodes: 4,5,8,10-12	3 switches, Nodes: 0,1,3	3 switches, Nodes: 9,13,14	4 switches, Nodes: 2,6,7,15	-

From the Figure 4.6, it is observed that latency actually decreases from k=2 to k=3 scenario, which is due to decrease in SC latency with increase in controller, while addition of a single controller does not increase much CC latency. Further addition of the controllers, i.e. k=4 and k=5, the CC latency increases significantly due to addition of control path between the controller while there is decrease in SC latency due to addition of the controller, thus overall increasing the global latency with increase in controller. When comparing NMR with BAT, the global latency of NMR is slightly better than BAT algorithm in scenario when 90% priority is given to SC latency, while 10% priority is given to CC latency.

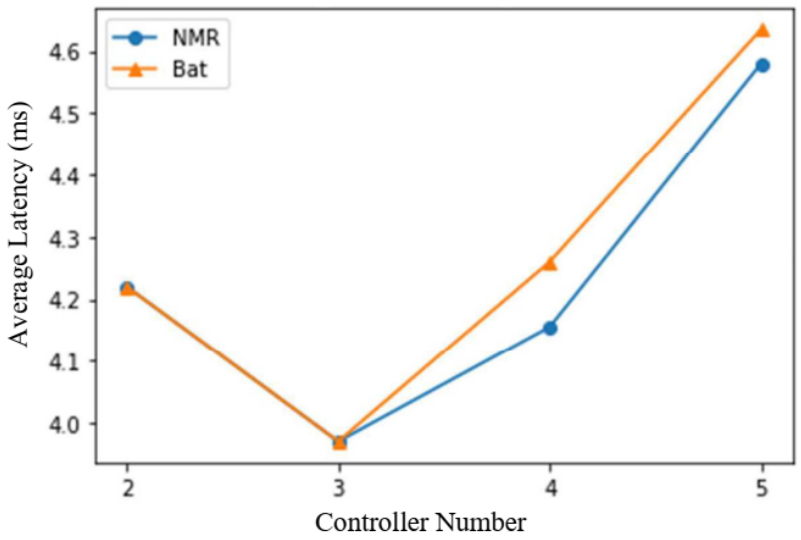


Figure 4.6: Global average latency with increasing controller at 0.9 SC + 0.1 CC in Ernet topology.

II. For Ernet topology with 0.8 SC and 0.2 CC:

In this scenario, 80% priority is given to SC latency and 20% priority is given to CC latency. For k=2-5 both NMR and BAT algorithm have the same placement of controller and thus have same global average latency which is summarized in Table 4.7.

Table 4.7: Ernet: Controller placement for K=2 and 3 using both NMR and BAT

Number of controllers (K) at 0.8SC + 0.2CC	**Number of controlled switches**			
	Node(8)	**Node(0)**	**Node(3)**	**Node(7)**
2 (NMR/BAT) algorithm	9 switches, Nodes: 4,5,8-14	7 switches, Nodes: 0-3,6,7,15 switches	-	-
3 (NMR/BAT) algorithm	9 switches, Nodes: 4,5,8-14	-	3 switches, Nodes: 0,1,3	4 switches, Nodes: 2,6,7,15

Due to the same placement of the controllers in this scenario, both NMR and BAT algorithms have same global average latency as shown in Figure 4.7. When the controller is increased from k=2 to k=3, the global average latency has increased by 8%, while for scenario when additional controller is added, i.e. k=4, the average latency increases by almost 14% and when there are 5 controllers, there is increase in latency by almost 19% compared to 4 controllers scenario. The observed increase in latency is due to additional path cost due to increase in number of controllers. Whenever, a new controller is added, the shortest path from all the previous controllers needs to be assessed and calculated, which ultimately contributes to increase in global average latency.

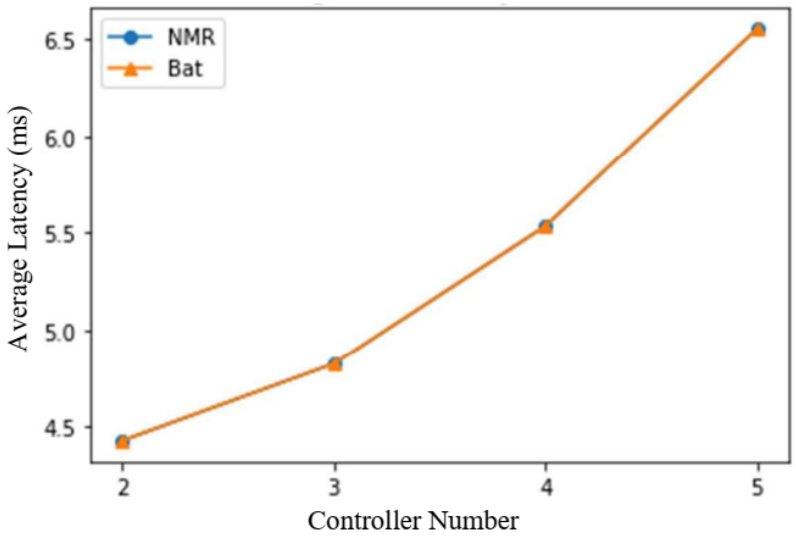


Figure 4.7: Global average latency with increase in number of controllers at 0.8 SC + 0.2 CC scenario in Ernet topology.

A comparison of different case scenarios has been summarized in Table 4.8 and presented in Figure 4.8 for different controllers.

Table 4.8: Global average latency variation with different cases for SC and CC using NMR and BAT algorithms

Cases	Average SC latency	Global average latency	Remarks
I. Only SC	Decreasing with increasing controller number	-	-
II. 0.9SC+0.1CC	-	Not decreasing with increasing controller number	Slight decline at k=2 and 3 and Slight increment after after k=4
III. 0.8SC+0.2CC	-	Starts rising with increasing controller number	Effect of increase in CC latency

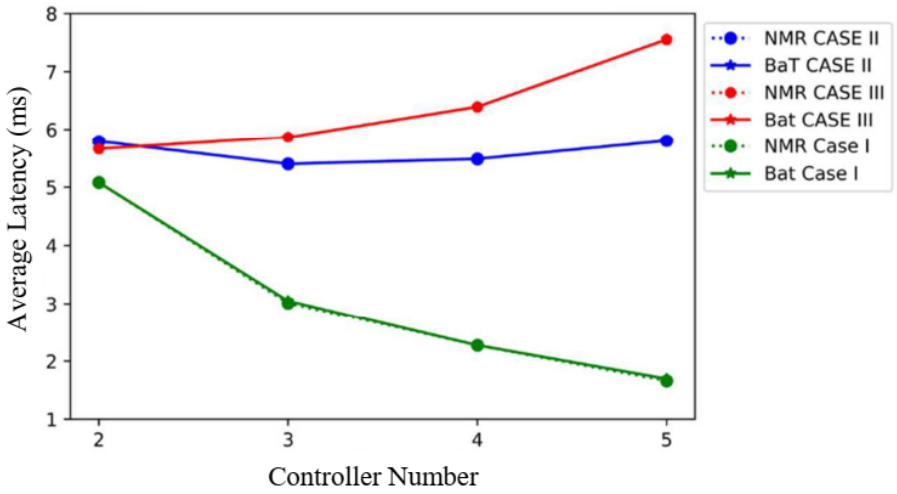


Figure 4.8: Global average latency variation in different cases of the NMR and BAT algorithms

4.1.4 Execution time complexity of algorithms

Execution complexity refers to time taken by algorithms to find the best placement of the controller. The Figure 4.9 illustrates that with an increase in the number of controllers, and hence the placement matrix in the algorithm increases. Due to this increase in the placement matrix, the possible combinations of the placement increase, which results in complexity of algorithms. Here, the NMR algorithm performs slightly better. When comparing two topologies, with an increase in number of nodes of the Savvis topology, the execution time of Savvis topology is higher than that of Ernet topology.

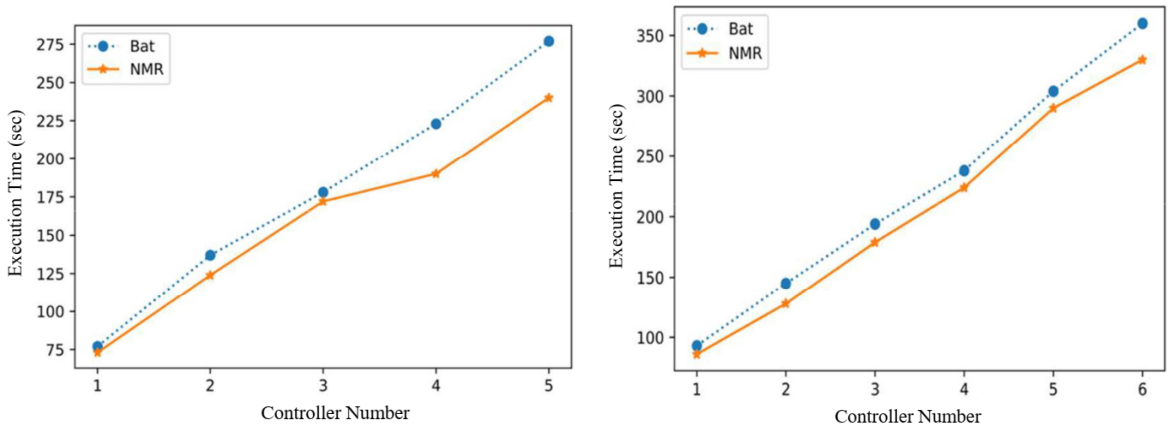


Figure 4.9: Execution time Complexity of algorithms with increase in number of controller in (a) Ernet and (b) Savvis topology

4.2 Traffic Driven Load Balancing

4.2.1 Dataset pre-processing

In order to train ML models for real-time classification in the future, the dataset generated during the network's data collecting phase must first undergo pre-processing. After normalizing all three traffic types to the appropriate number of samples (i.e., 60,036), which reduces the chances of over-fitting to a particular traffic type. In this model, the feature importance score of the sixteen features is estimated first using a correlation matrix. Based on this score, the top eight features, which are shown in the Table 4.9, are selected in order to simplify the next steps and expedite the classification process.

Table 4.9: Top 8 features for traffic classification based on feature importance

No.	Name	Specification
1	Forward Average Bytes per second	0.40223
2	Forward Average Packets per second	0.27834
3	Reverse Average Packets per second	0.10243
4	Reverse Average Bytes per second	0.08076
5	Delta Forward Bytes	0.05202
6	Forward Instantaneous Bytes per Second	0.04397
7	Delta Forward Packets	0.01610
8	Forward Instantaneous Packets per Second	0.01186

All subsequent stages of training a classification model employ these attributes. The removal of unnecessary elements from the feature set facilitates efficient model training, resulting in accurate real-time classification of all traffic classifications.

4.2.2 Comparison of various classification model

Different classification models are tested on the dataset to identify which one best classifies the generated dataset and may be further implemented for real-time classification. Table 4.10 displays the accuracy, precision, recall and F1 score, but the weighted F1 score need not be necessarily equal to the harmonic mean of precision and recall.

To justify the Table 4.10 and show the detailed calculation of the F1-score, it is shown with an example of a decision tree. The confusion matrix of the decision tree is shown in Fig. 4.10.

Table 4.10: Comparison of various traffic classification model

ML Algorithm	Accuracy (%)	Precision (%)	Recall (%)	F-1 Score (%)
Logistic Regression	65	82	65	64
SVM	65	83	65	65
K-NN	56	81	56	52
XGBoost	66	79	66	66
Decision Tree (CART)	70	84	70	70
OPTICS	29	60	29	23

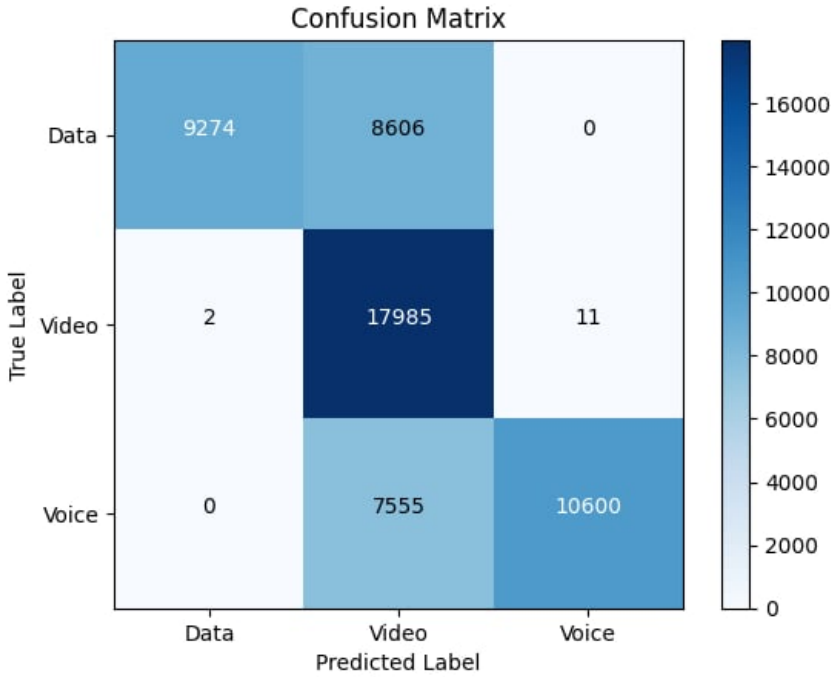


Figure 4.10: Confusion Matrix of Decision Tree

Accuracy = (Corrected prediction / Total prediction) = (9274 + 17985 + 10600 / 54033) = 0.7006

Calculating Precision, Recall and F1-score for each class metric

Class: Data

TP = 9274

FP = 2 (from video) + 0 (from voice)=2

FN = 8606

Support = 17880

Precision (Data) = 9274 / (9274 + 2) ≈ 0.9998
Recall (Data) = 9274 / (9274 + 8606) ≈ 0.5187
F1 (Data) = 2 × (0.9998 × 0.5187) / (0.9998 + 0.5187) ≈ 0.6826

Class: Video
TP = 17,985
FP = 8606 (from Data) + 7555 (from Voice) ≈ 16,161
FN = 13 (2 + 11)
Support = 17,998
Precision (Video) = 17985 / (17985 + 16161) ≈ 0.5264
Recall (Video) = 17985 / (17985 + 13) = 0.9993
F1 (Video) = 2 × (0.5264 × 0.9993) / (0.5264 + 0.9993) ≈ 0.6895

Class: Voice
TP = 10600
FP = 11 (from Video) + 0 (from Data) = 11
FN = 7555
Support = 18,155
Precision (Voice) = 10600 / (10600 + 11) ≈ 0.9989
Recall (Voice) = 10600 / (10600 + 7555) ≈ 0.5839
F1 (Voice) = 2 × (0.9989 × 0.5839) / (0.9989 + 0.5839) ≈ 0.7358

Weighted precision = (17880 / 54033)× 0.9998 + (17998 / 54033) × 0.5264 + (18155 / 54033) × 0.9989 ≈ 0.8417
Weighte Recall = (17880 / 54033) × 0.5187 + (17998 / 54033) × 0.9993 + (18155 / 54033) × 0.5839 ≈ 0.7007
Weighted F1-score = (17880 / 54033) × 0.6826 + (17998 / 54033) × 0.6895 + (18155 / 54033) × 0.7358 ≈ 0.7026

Also, the Figure 4.11 shows that for the generated dataset, the Decision Tree (CART) classifier achieves the best classification results, whereas OPTICS has the worst classification report. Furthermore, it can be seen that logistic regression yields good results for all of the parameters.

After classification, 65% accuracy rates for LR and SVM are achieved, respectively.

The program performs well in regard to accurately classifying voice and data traffic in both cases, but it performs poorly in regard to video traffic.

Compared to the first two algorithms, KNN has a different pattern. It accurately classifies data and video traffic, while voice traffic is only classified with a precision of 43%. K-NN's overall accuracy is likewise quite low, coming in at only 56%. Both data and voice traffic are accurately classified by XGBoost, however video traffic is not.

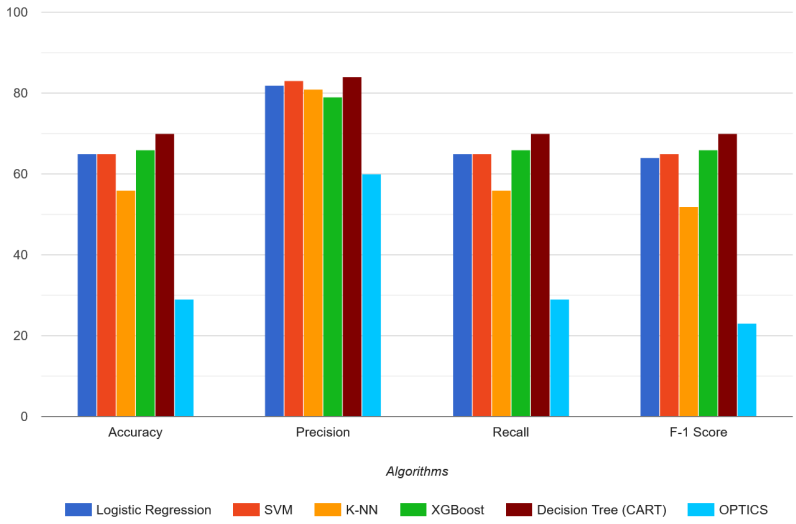


Figure 4.11: Evaluation of various algorithms on the training dataset

During classifying, CART performs accurately; however, it cannot accurately classify video traffic as logistic regression and SVM. However, the total F1 score of the decision tree is higher than any other algorithm's, and its accuracy is higher than average at 70% (still needs to be improved). Due to its strong classification report and strong performance in terms of accuracy, precision, recall, and f-1 score, Decision Tree is the most preferred algorithm for classification. The decision tree classifier is also well suited for real-time classification due to its short execution time.

4.2.3 Validation of classification model

During the model training phase, validation data are utilized to adjust the model's parameters, or hyper parameters, to prevent overfitting or underfitting. It offers a means of evaluating the model's performance while it is being trained, enabling modifications.

Here, the validation dataset is not generated in the environment. Instead, a publicly available dataset[1] is used for validation to validate if the above result for all ML algorithms is not over-fitted to a particular dataset. The validation dataset uses more or less similar features set as used in the generated dataset, so the mentioned dataset is used for validation of the classification model. This dataset consists of 87 features. However, only 8 features used in the training dataset are extracted and used for validation purposes as well to ensure similarity with the training dataset.

The same algorithms with the same parameters are tested on a validation dataset to determine if the results for all machine learning methods are not over-fitted to a particular data set. The sample of the validation dataset is shown in Table 4.11.

Table 4.11: Sample of validation dataset

Flow Duration	Total Fwd Packets	Total Back-ward Packets	Fwd Packet Length Mean	Total Length of Fwd Packets	L7 Protocol
45523	22	55	6.000000	132	131
1	2	0	6.000000	12	131
217	1	3	0.000000	0	7
78068	5	0	215.200000	1076	131
105069	136	0	2305.544118	313554	131

The comparison graph of all algorithms implemented for classification of validation dataset is shown in figure 4.12.

The comparison graph for each algorithm for the validation dataset also reveals results that are nearly identical to those of the generated dataset. Here, logistic regression produces the best results. However, the decision tree (CART) method also yields results that are almost identical; hence, based on both charts, CART is regarded as the best algorithm. The OPTICS algorithm's output is not included in the validation dataset because it is unable to produce it even after running for 60 minutes and eventually experiencing an execution timeout.

4.2.4 Comparison of execution times of classification models

Figure 4.13 compares the execution times of all methods for the generated dataset and the validation dataset.

[1]https://www.kaggle.com/datasets/jsrojas/ip-network-traffic-flows-labeled-with-87-apps/data/

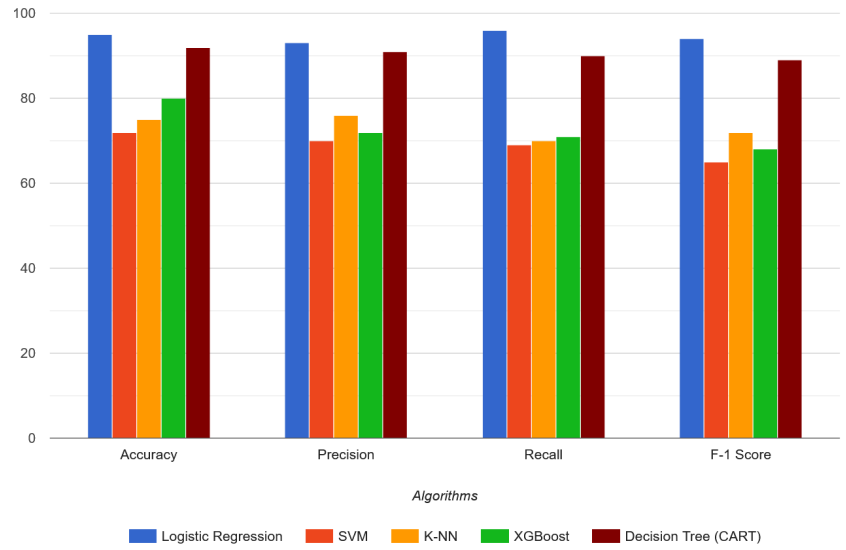


Figure 4.12: Evaluation of algorithmic output for validation dataset

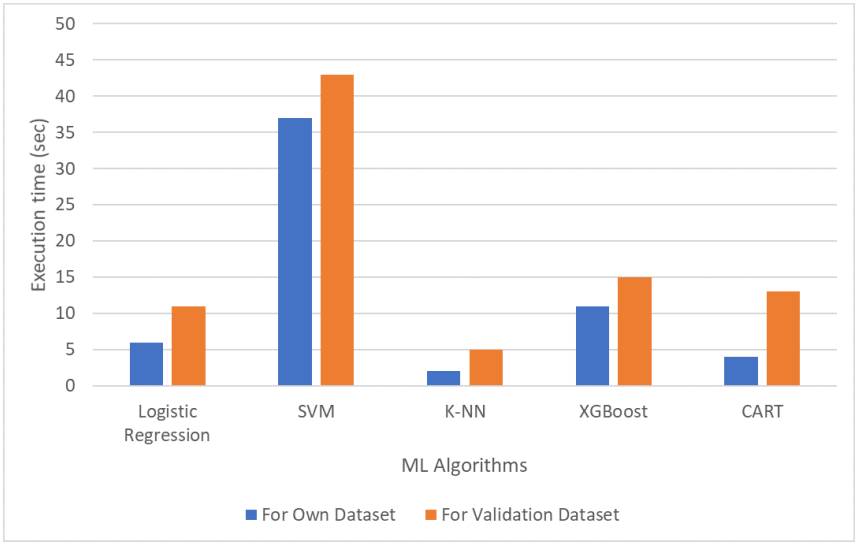


Figure 4.13: Execution time for various algorithms in classification

From the graph, it is clearly visible that K-NN has lowest execution time for both the generated and validation dataset. Since K-NN is an instance-based approach it does not require a training phase in which a model is learned from training data. Rather, it just stores the training data and uses it to generate predictions during testing. K-NN can predict results quickly because it simply has to calculate the distance between the test instance and the saved training instances.
Additionally, among the five methods, SVM takes the longest to execute. Due to the complexity of the training process, SVMs require a training phase in order to identify the optimum decision boundary that separates the various classes, which increases the execution time of SVMs.

Once again, CART is identified as the best algorithm for real-time traffic classification when execution time and the classification report are taken into account. Logistic regression is an other algorithm to be considered in mind.

OPTICS is not included in the graph since its execution took longer than a minute for the generated dataset and caused a "runtime timeout" for the validation dataset.

4.2.5 Dynamic load balancing and QoS improvement

For testing the impact of load balancing in multi-controller SDN environment, ten flows in Fig 4.14 are sent across the hosts in four case scenarios: no load balancing, traffic classification-based load balancing with a single controller, two controllers and three controllers.

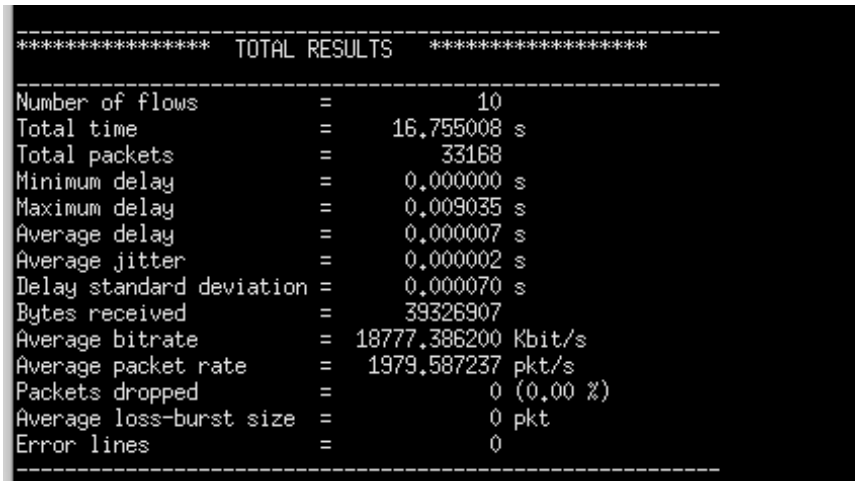


Figure 4.14: Ten flows parameters

The detail is demonstrated in Table 4.12. When compared to the scenario with a simple switching implementation or no load balancing, the scenario with the implementation of the traffic-based load balancing module shows improvement in a number of parameters.

To understand the impact on bit-rate and packet rate after application of load balancing module in the Ryu controller, a graph is generated for ten flows from host 1 to host 2.

From the Figures 4.15 and 4.16, The results indicate that the use of traffic classification-based load balancing improves the bit-rate and packet-rate of traffic flow on a network. When three controllers are used for load balancing instead of two, there is a noticeable increase in bit rate and packet rate, which justifies adding more controllers to increase the bit rate of the entire network.

Table 4.12: Comparison of load balancing scenarios on same network

Condition	Path	Flows	Jitter (ms)	Bit-rate (bits/s)	Packet-rate (packets/s)
No load balancing	H1-H3	6	159	5317	23
	H3-H1	8	149	5303	23
	H1-H2	10	47	11065	252
Load balancing with single controller	H1-H3	6	155	5317	26
	H3-H1	8	155	5316	28
	H1-H2	10	18	12266	426
Load balancing with two controllers	H1-H3	6	99	5429	26
	H3-H1	8	104	5418	30
	H1-H2	10	7	16822	982
Load balancing with three controllers	H1-H3	6	98	5699	49
	H3-H1	8	105	5690	49
	H1-H2	10	6	18884	1969

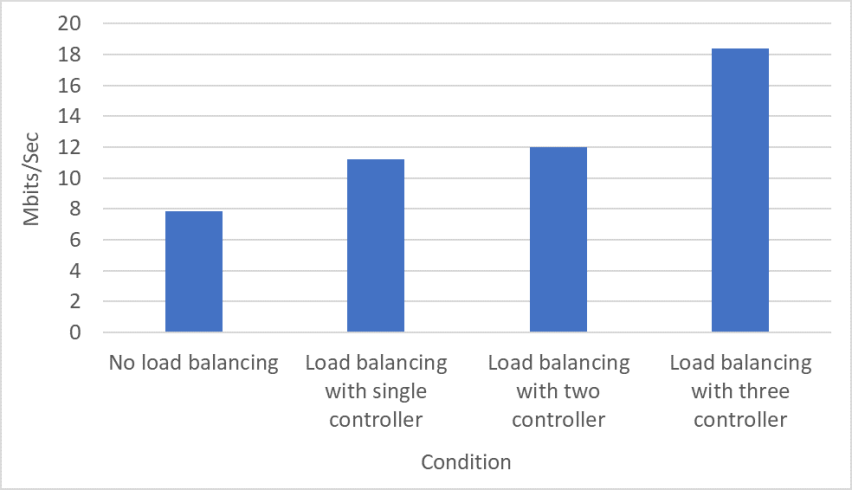


Figure 4.15: Comparison of bit-rates for different conditions

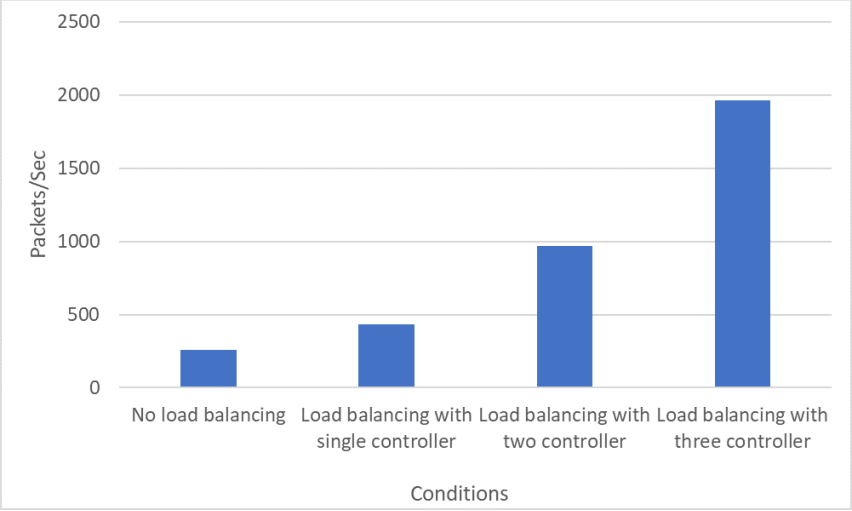


Figure 4.16: Comparison of packet-rates for different conditions

Here, both bit-rate and packet-rate output demonstrate that increasing the number of controllers in the network considerably enhances throughput, which is a critical factor for real-time traffic in any type of network.

Finally, jitter also monitors the variation in packet latency, or the interval of time between a signal's transmission and reception as it travels through the network. Network congestion, poor hardware performance, varying network speeds, and a lack of packet priority are common causes of jitter. In general jitter will increase as a signal flows through multi nodes but improved load balancing can decrease network jitter, which will improve the quality of service for real-time applications like voice and video. The Figure 4.17 illustrates the comparison of average jitter for load balancing and non-load balancing scenarios. In the figure, jitter is slightly decreased when single controller-based load balancing is performed. Although the graph shows that jitter appears to have increased at one point relative to no load balancing, this could be due to the large volume of packets queuing to be classified by a single controller.

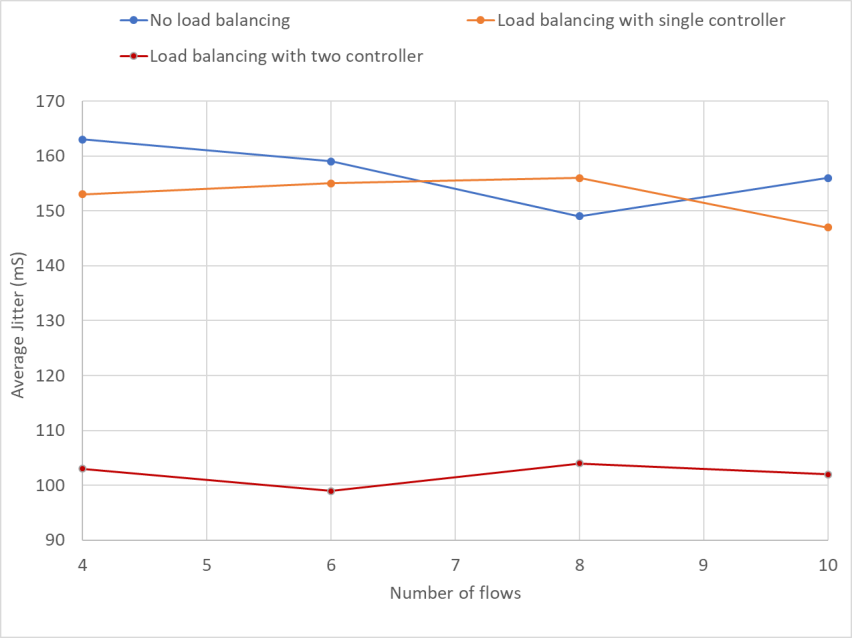


Figure 4.17: Comparison of average jitter for different conditions

The reduction in jitter confirms the improvement in QoS of network by implmenting traffic classification based load balancing in multicontroller SDN environment. According to Table 4.12, the jitter in this research work is nearly identical for two controller-based and three controller-based implementation. This may be the result of the fact that, despite the presence of two controllers, neither controller experienced congestion during this testing, resulting in the same jitter result for both

circumstances.

4.3 Multi-Controller Placement Optimization with Attack Detection and Mitigation

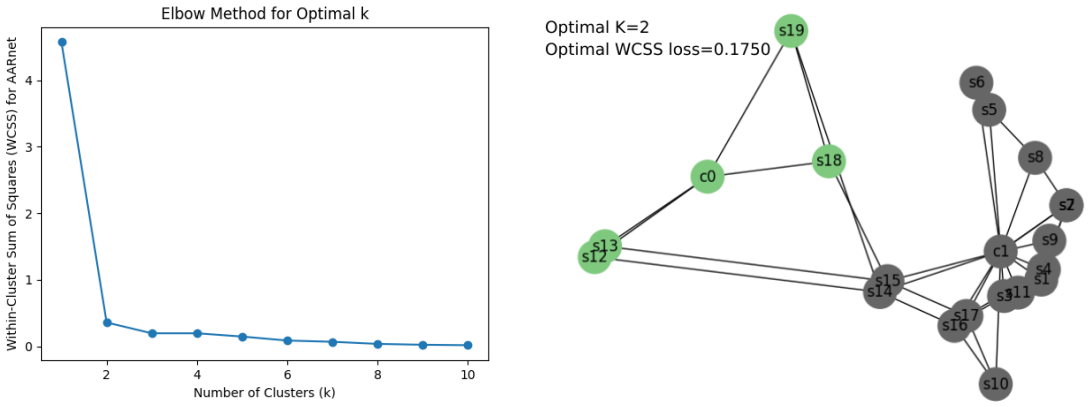
4.3.1 Algorithm performance for optimum multicontroller placement in Aarnet

K-means++ for Aarnet topology

For an unsupervised algorithm, determining the optimal number of clusters is a fundamental step. Elbow method identifies the optimal value of centroids (k) for K-means++ Algorithm. Continuous iteration for k=1 to k=10 is done, and for every value of 'k', WCSS is calculated . It provides the sum of square distances between each location (data point) and the centroids.

Figure 4.18a, which resembles an elbow, shows a graph of k versus its WCSS value. To identify the optimal number of clusters, the value for'k' must be selected at the "elbow" point, this is the point where the distortion starts decreasing in a linear fashion [176].

For k=2, two controllers, c0 and c1 are placed, yielding a WCSS loss of 0.1750, as shown in Figure 4.18b.



(a) (b)

Figure 4.18: K-means++ result for Aarnet. (a) Elbow curve for Aarnet. (b) Optimal Aarnet topology with K-means++, optimal k=2.

OPTICS for Aarnet topology

Running OPTICS on Aarnet yielded similar configuration as produced by K-means++, with two clusters and WCSS loss of 0.2012. Hence, two controllers c0 and c1 are placed as shown in Figure 4.19.

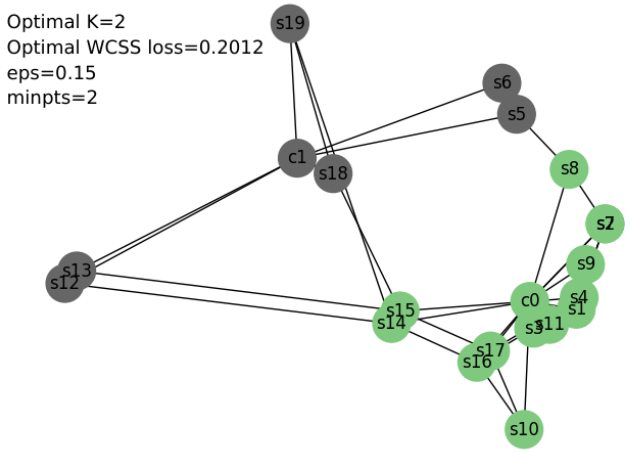


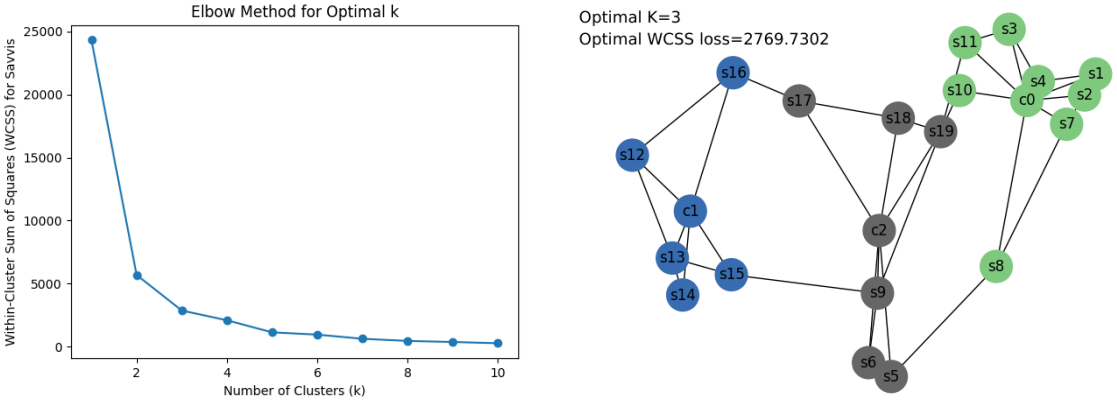
Figure 4.19: Aarnet Topology via OPTICS.

Inference: Since, (WCSS Loss of K-means++) < (WCSS Loss of OPTICS), result from K-means++ was implemented for Aarnet yielding 2 clusters.

4.3.2 Algorithm performance for optimum multi-controller placement in Savvis

K-means++ for Savvis topology

Similarly, running K-means++ in Savvis resulted in an elbow at k=3 with loss of 2769.7302 as shown in Figure 4.20a. Hence,controllers c0, c1,and c2 are placed accordingly as shown in Figure 4.20b.



(a) (b)

Figure 4.20: K-means++ result for Savvis. (a) Elbow curve for Savvis. (b) Optimal Savvis topology with K-means++, optimal k=3.

OPTICS for Savvis topology

Again, running OPTICS on Savvis also yielded configuration with three controllers as depicted in Figure 4.21 with WCSS loss of 2760.2634.

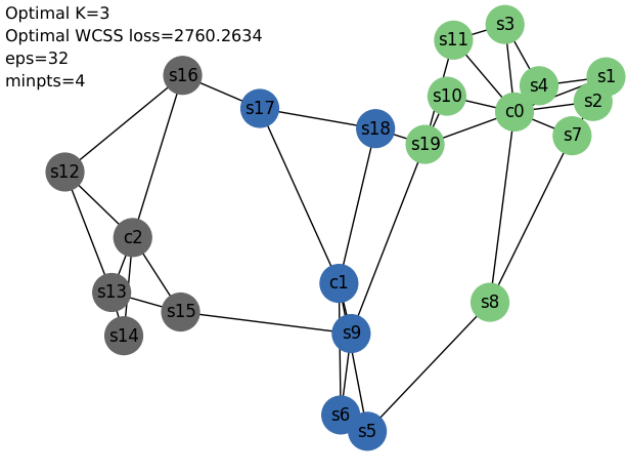


Figure 4.21: Savvis Topology via OPTICS.

Inference: Since (WCSS Loss of OPTICS) < (WCSS Loss of K-means++), the result of OPTICS was implemented for Savvis, yielding 3 clusters.

The WCSS in Savvis is relatively high compared to Aarnet. This difference can be attributed to variation of link speeds in these networks. In Aarnet, the link speeds are typically in the order of Gigabits per second (Gbps), whereas in Savvis, they range in the order of Megabits per second (Mbps). This factor accounts for a larger

denominator in Aarnet compared to Savvis in Equation (3.19), resulting in a smaller distance for Aarnet.

4.3.3 System design scenario

After controller placement optimization, Figures 4.22, 4.23 represents the resulting system diagram incorporating controllers, switches, and the central IDS along with flow of actions. The arrows show the direction and type of packets flowing in the environment whereas the lines represent the physical connections between the devices.

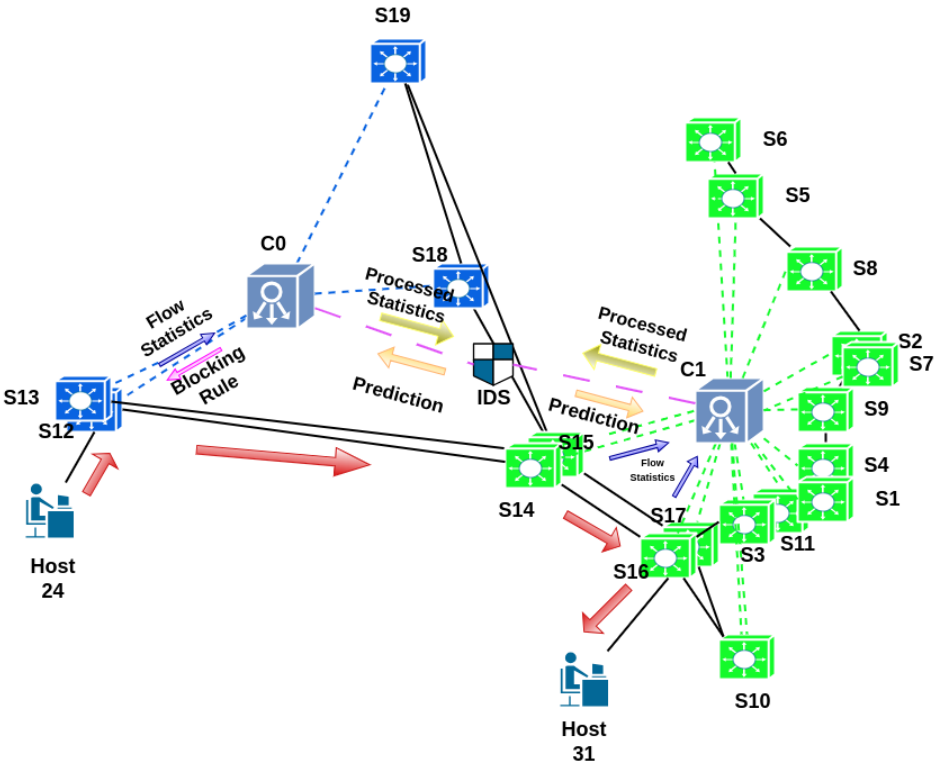


Figure 4.22: System diagram of Aarnet topology (resulted from Kmeans++).

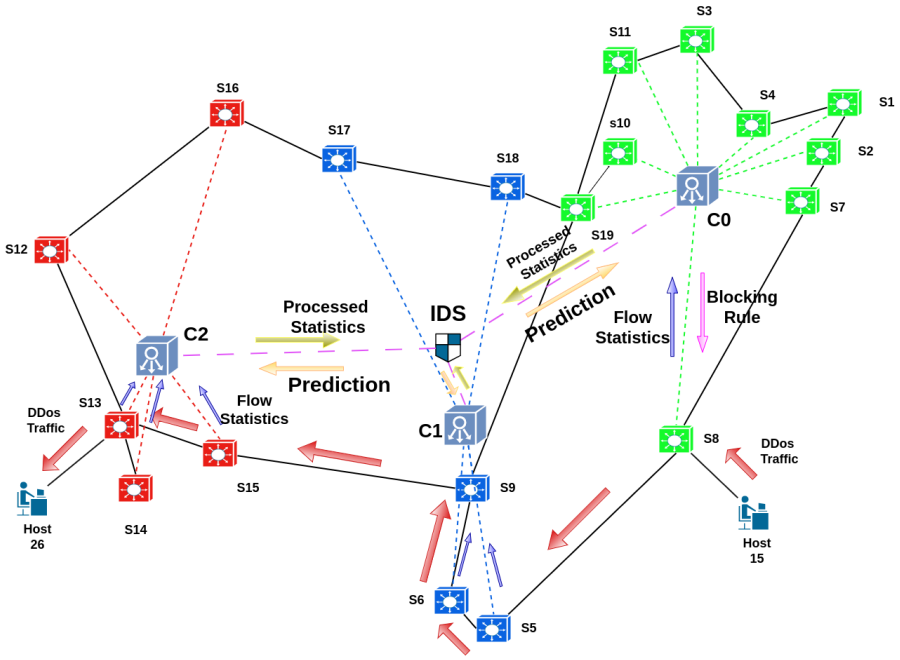


Figure 4.23: System diagram of Savvis topology (resulted from OPTICS).

Meanwhile, the system can detect and mitigate attacks appearing from multiple hosts simultaneosly. For simplicity and clarity, an attack from only one host is illustrated in Figure 4.22, 4.23. DDoS is used despite a single host due to the host using multiple spoofed IPs during the attack.

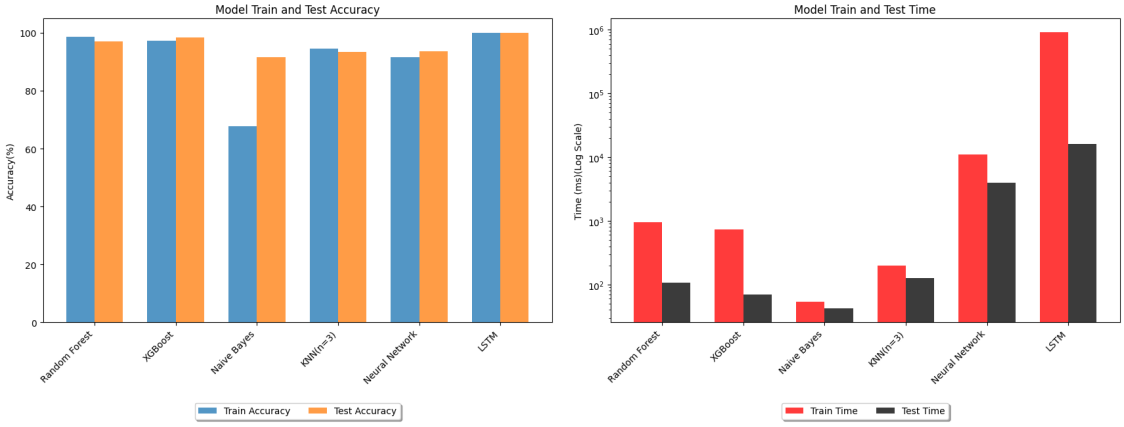
4.3.4 Performance evaluation of ML models

Computation time and accuracy are key factors that define the performance of an (IDS). Therefore, the performance of ML models is assessed based on test time, which refers to the time taken to process the entire test set, as well as their accuracy, including precision, recall, and F1-score.

From Table 4.13, it can be observed that Random Forest, XGBoost and LSTM offers >95% test accuracy, this solves detection part, however, to perform swift mitigation, quicker response is equally important, and this faster response time has set XGBoost apart from others. Here, XGB offers both optimum balance of test-accuracy (98.5%) and test-time (70 millisec). Hence, XGBoost is implemented in IDS for detection of DDoS attacks.

Table 4.13: Performance in terms of Train-Test Time and Train-Test Accuracy in terms of percentage

Model	Train Time	Test Time	Train Acc.	Test Acc.
Random Forest	946ms	106ms	98.65	97.00
XGBoost	728ms	70ms	97.3	98.5
Naive Bayes	54ms	42ms	67.7	91.53
KNN(n=3)	198ms	126ms	94.51	93.32
NN	11.6s (50epochs)	4s	91.62	93.6
LSTM	15m (15epochs)	16s	99.9	99.9



(a) (b)

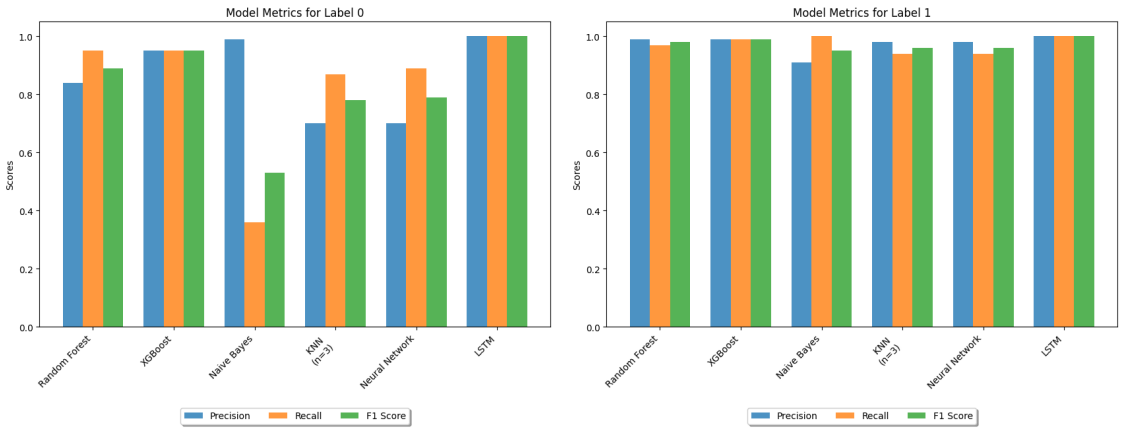
Figure 4.24: Performance of different models. (a) Train and Test Accuracy. (b) Train and Test Time.

Naive Bayes has significantly lower train accuracy as indicated in Figure 4.24a, also in Table 4.14, very low recall rate (TPR) is observed, so there is no proceed with Naive Bayes.

KNN shows comparatively higher test time (Table 4.13)/(Figure 4.24b) and lower recall rate (TPR) (Table 4.14)/(Figure 4.25a) for Benign traffic. The NN model yields an accuracy of 93.6% when used with the generated DDoS dataset. Despite of high test accuracy, NN has lower Recall for Benign (label 0) traffic. Also, it shows higher test time of 4 sec.

Table 4.14: Classification report of algorithms.

Model	Label	Precision	Recall	F1 Score
Random Forest	0	0.84	0.95	0.89
	1	0.99	0.97	0.98
XGBoost	0	0.95	0.95	0.95
	1	0.99	0.99	0.99
Naive Bayes	0	0.99	0.36	0.53
	1	0.91	1.00	0.95
KNN (n=3)	0	0.70	0.87	0.78
	1	0.98	0.94	0.96
Neural Network	0	0.70	0.89	0.79
	1	0.98	0.94	0.96
LSTM	0	1.0	1.0	1.0
	1	1.0	1.0	1.0



(a) (b)

Figure 4.25: Performance graph of ML models. (a) Metrics for label 0 (b) Metrics for label 1.

Furthermore, in Table 4.14, although both algorithms exhibit identical precision for the label 1 (DDoS traffic), XGBoost demonstrates a higher recall for DDoS attacks than Random Forest. This indicates that, among all actual positive instances of DDoS attacks, XGBoost can predict a greater number of attacks successfully. Consequently, XGBoost achieves a higher F1 score, reflecting a better balance between precision and recall.

The Receiver Operating Characteristics (ROC) Curve depicted in Figure 4.26 reveals findings that support Table 4.14. Better classifying algorithms have strong inclination towards the top left corner. Complex deep learning models such as

LSTM outperforms other models. But despite the best performance, for this case, it took significantly high test time of 16 sec to classify the same data compared to XGBosst which took 70ms. The high testing time is proportional to the decision time of the IDS. If LSTM had been implemented, it would have resulted in a higher response time due to the time-series nature and sliding-window methodology, which adds additional computational overhead. In addition, every LSTM value looks 1 in the algorithm classification report in Table 4.14. Therefore, not suitable for near real-time detection and mitigation. Consequently, LSTM is not implemented in this model.

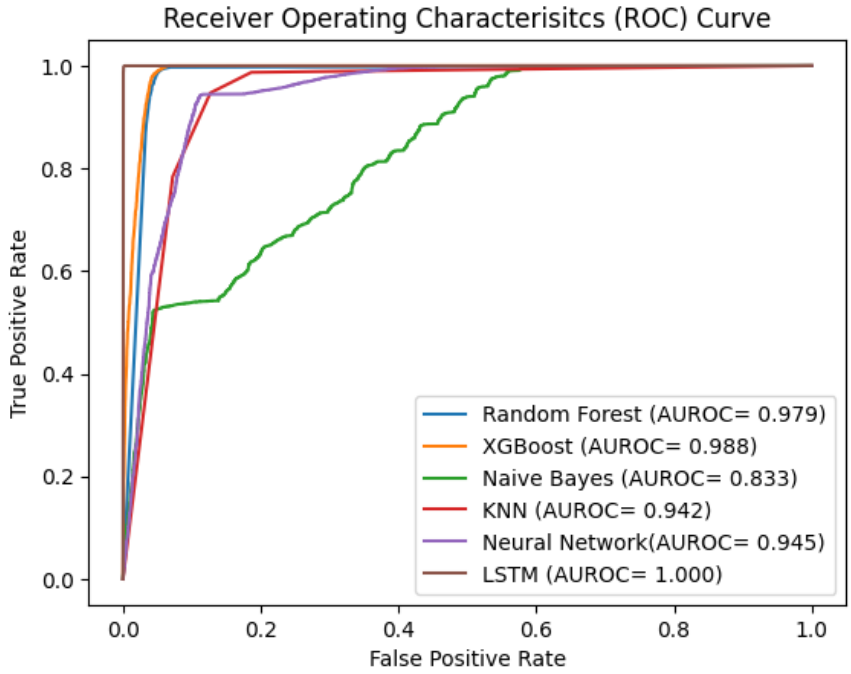


Figure 4.26: ROC curve

From Table 4.15, it can be concluded that this work shows a significant improvement in multicontroller DDoS detection. Although several studies, such as Clinton et al. [177] and Ai-Dunainawi et al. [178] achieve comparable accuracy, precision, and recall using single-controller setups, the LSTM model with 99.9% test accuracy, have achieved state-of-the-art performance. Specifically, the InSDN dataset from Clinton et al. [177] is used, as it best represents the SDN environment, although other datasets are considered in their paper.

Furthermore, here, the optimal XGBoost model achieves an accuracy that is only 1.49% lower than the best LSTM models. This shows that despite focusing on both detection and mitigation while considering both accuracy and latency, this approach

delivers competitive performance compared to models that prioritize accuracy alone. Additionally, the dataset here differs from those of the mentioned models, as it is the first dataset collected in a multi-controller environment and contains different feature sets, marking a considerable contribution to the field of SDN DDoS security.

While the comparison metrics (Accuracy, Precision, Recall) of the mentioned works in the Table 4.15 are very similar, the major contrasting factor of this work is the setup of the SDN environment. The setup of multi-controllers and a primary dataset, differing from other research, for the ML model provides a more closer resemblance to real world scenarios. Despite of a complicated setup of a multi-controller environment with a central IDS trained using a primary dataset obtained from SDN itself, the performance metrics manages to stand on par with the recent works on SDN DDoS security.

Table 4.15: Comparison with existing works on DDoS detection.

Author	Domain	Dataset	Accuracy	Precision	Recall
Clinton et al. [177]	Single Controller	InSDN Dataset	99.74	99.74	99.74
Al-Dunainawi et al. [178]	Single Controller	Custom Dataset	99.99	99.99	99.99
Hiris et al. [179]	Single Controller	Novel Dataset	98.97	98.33	96.37
Gebremeskel et al. [153]	Multi controller SDN	CIC DDoS SDN 2019 (Flow based)	99.42	99.40	99.464
Proposed model	Optimally Placed Multi-controller	Flow Table based Custom Dataset	98.50	97.00	97.00

The notable thing in this work is the use of simpler attacks (ICMP, TCP SYN, UDP flood attacks), which could have been easily resolved by simpler non-ML algorithms like entropy methods, bandwidth probes, hybrid statistical methods, etc. The ML-based approach is done to exploit its learning capabilities. Traditional algorithms that are mostly threshold-based seem to need manual tuning to adapt to different rates of attacks. Hence, to eliminate the necessity of any manual tuning, the ML approach is utilized, which is also able to detect attacks efficiently. Additionally, it is aimed to establish a base framework of ML on this flow table-based dataset in a multi-controller environment. Detection of advanced attacks like LDDoS, ping of death, and application layer DDoS are kept as future works on this new framework.

Finally, compared to a single-controller setup, a multi-controller SDN setup is more complex. In the case of a single controller scenario, for an attack mitigation action, both the source and victim are present inside the domain of the same controller. This setup a simple detection and mitigation algorithm can be executed on the single controller (simple algorithm in the sense that with a lesser number of controllers involved, there will be fewer things to take care of and the use cases are also relatively narrow for practical cases). However, when either of an attacker or a victim lies in a separate domain, single controller algorithms cannot be employed because a coordinating mechanism is required between controllers. To address this problem, we need a more advanced mechanism where attacks from different controller domains can also be handled. For this, Algorithm 7 and Algorithm 8 are implemented in a multi-controller environment. This setup, naturally forms a complex network with the necessity of a more advanced algorithm, since data from domains of multiple controllers should be looked at while making decision and multi-controllers should also be in correlation (via IDS) such that mitigation is performed by precisely locating the attacker. Furthermore, controller placement considerations are also important while dealing with multi-controller environments adding extra complexity.

4.3.5 Attack detection and mitigation

After XGBoost is deployed in both Savvis and Aarnet networks. Detection and mitigation algorithm are verified by a case with and without mitigation module. In presence of DDoS attack it is expected to have high number of packets flowing through the victim. Without the mitigation module in action, a high number of packets from the DDoS attack are being sent to the victim, evident from Figures 4.27 and 4.28.

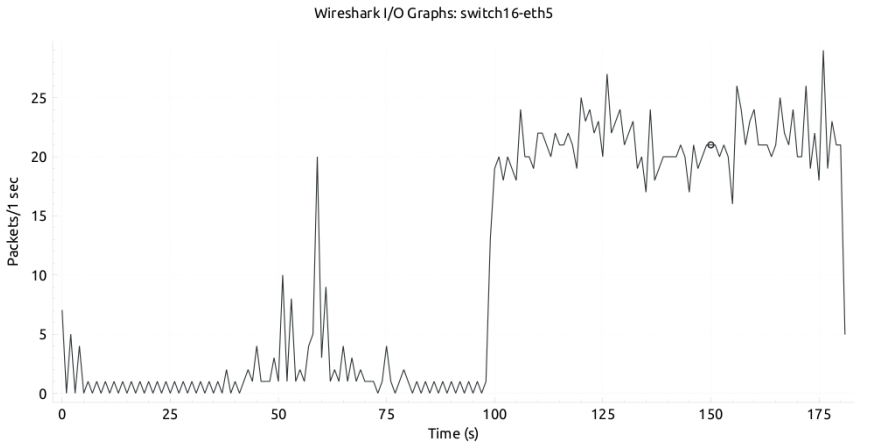


Figure 4.27: Aarnet network without mitigation.

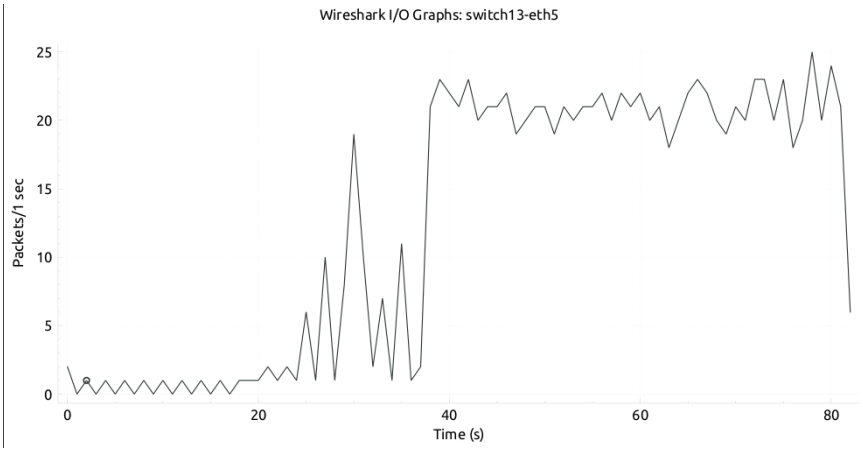


Figure 4.28: Savvis network without mitigation.

With the Mitigation Module in action, there is a significant reduction in the number of packets, as shown in Figures 4.29 and 4.30, thereby preventing the attack. Also, from Figure 4.31, it can be observed that CPU utilization increases significantly at the instance of DDoS attack (from time 12:51:00-12:51:30), and after the application of mitigation module, it returns back to a normal state.

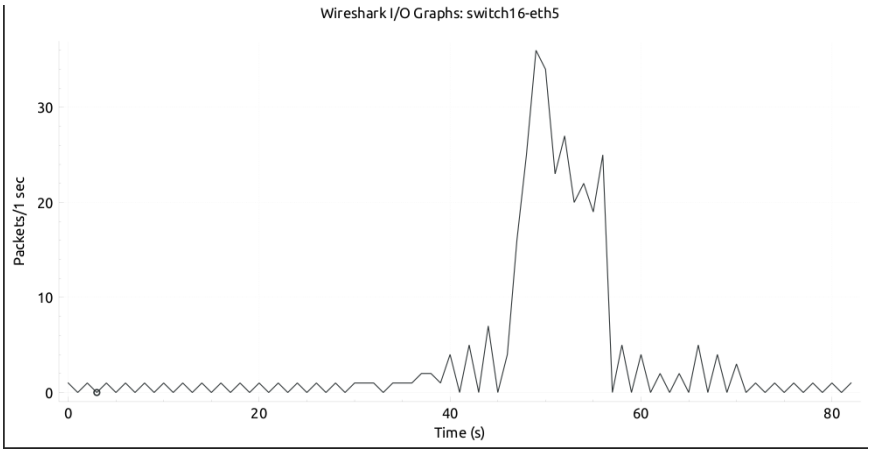


Figure 4.29: Aarnet network with mitigation.

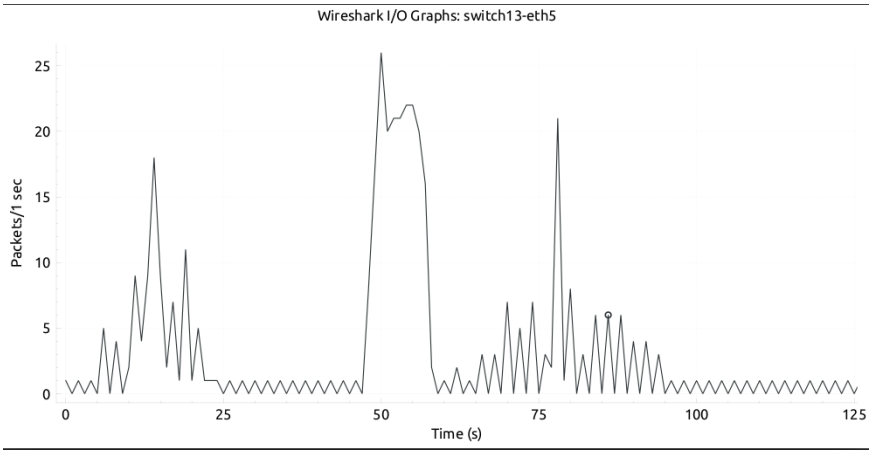


Figure 4.30: Savvis network with mitigation.

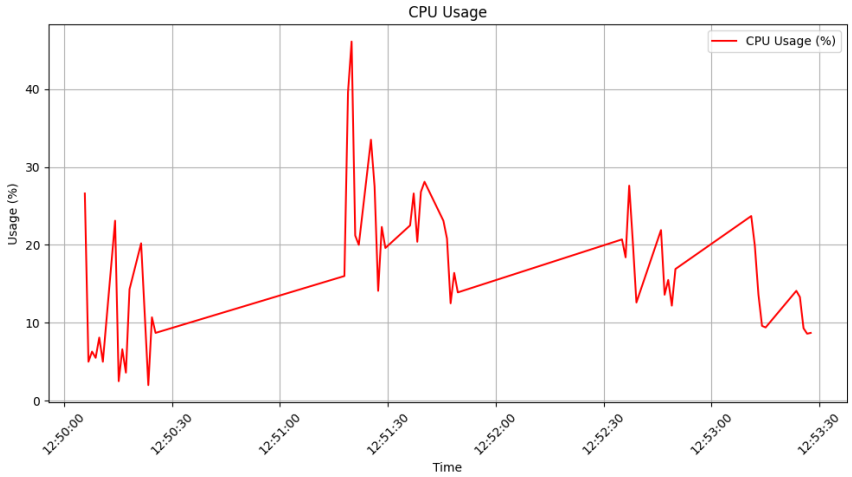


Figure 4.31: CPU utilization with mitigation

Figures 4.29, 4.30, and 4.31 demonstrate that the model effectively detects and mitigates attacks in multi-controller environments. Additionally, normal service is re-established as soon as the attack is over. Figures 4.29 and 4.30 illustrate that, aside from the largest spike, approximately from time 41 to 59 in Aarnet and from time 49 to 58 in Savvis, there is normal packet exchange. After the attack ends, the network continues with packet exchanges, as indicated by the smaller/thinner spikes that appear afterward as the largest/broadest spike. This detection and mitigation addresses the limitations of previous models, which are either specific to certain topologies or focus solely on attack detection without considering network optimization. Furthermore, the algorithms show strong generalization across both Aarnet and Savvis topologies; the datasets generated from Savvis are used for testing in the Aarnet environment. This also confirms the robustness of the models.

This work mainly focuses on end host to host attacks. For the cases of attack against a controller, the flow generated is treated similarly like an attack to a host, and can be detected and mitigated using the same previous approach and algorithms. But in the case of IDS, a host cannot launch a DDoS attack directly against it, because, in this setup, the IP address of the central IDS is not made accessible to the end hosts.

Detection and mitigation of attack is displayed in terminal as shown in Figure 4.32, displaying "ATTACK DETECTED" message along with both controller and switch number of attack source, and interface where blocking mechanism will be applied.

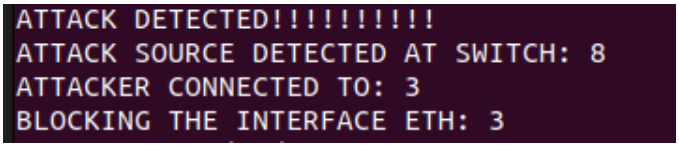


Figure 4.32: Message display in terminal for Detection and Mitigation.

4.4 Chapter Summary

Using the mathematical formulations provided in Chapter 3, all of the concepts and methods were developed and put into practice using simulations and programming. Subsequently, the experimental results were analysed as part of the research, and the work is appropriately represented in tables, graphs, and charts. Suitable MCPOs were presented for different cases: SC and CC latency in Ernet and Savvis topologies, respectively. Similarly, traffic classification and load balancing using ML have been performed, with dynamic load balancing and QoS improvements. Also, near real time attack detection and mitigation over MC-based SDN environment using ML has been evaluated, making a system design scenario considering the Aarnet and Savvis topologies.

Chapter 5

Conclusions and Recommendations

5.1 Conclusions

With the intention of establishing the context for SDN, a quick overview of SDN and its architecture is provided first. Research approaches related to CPP, MCPP, and load balancing techniques are discussed. To increase the effectiveness of SDN, controller placement must be done optimally. Multiple controllers are necessary for large-scale SDN in order to assure scalability and stability.

The analysis of the optimum controller placement considered in this study takes into account both the SC latency and the global latency (SC+CC latency). Three aspects—(i) effect on average SC latency; (ii) effect on global latency; and (iii) effect of algorithm execution complexity with an increase in controllers—are included in the analysis of the two algorithms, NMR and Bat, to find the optimal solution. The ideas and mechanisms are illustrated using two publicly available standard topologies, viz., Ernet and Savvis. The controller localization approach implemented with the NMR algorithm has slightly better results as compared with the Bat algorithm. This research is limited to finding the controller placement considering average SC latency and average CC latency.

Additionally, this study focuses on creating a model that can classify incoming traffic first according to its type and then use that information as a function for a load-balancing module to route the traffic through various paths for various classes of traffic. Different machine learning algorithms have been implemented for classification purposes. After comparing the outputs of different algorithms for the training dataset and validation data set, CART is considered to be the best algorithm.

In order to avoid network congestion and link failure, two backup controllers are implemented to handle traffic classification in the event that the main controller's CPU utilization exceeds the threshold value. The global controller keeps track of both the primary and secondary controllers. Further, the comparison of packet rate, bit rate, and jitter shows that applying load balancing based on traffic classification in multi-controller SDN environments enhances network performance as a whole.

Moreover, research on load balancing in SDN for data centres and ISP/Telco networks could cover a wide range of topics, including energy-efficient load balancing, intelligent multi-controller placement techniques, AI-based dynamic load balancing for multiple controllers, application-based load balancing, blockchain technology, and security issues in a multi-controller-based SDN environment.

Regarding SDN security, this study focuses on optimised multi-controller placement using Kmeans++ and OPTICS for real-time topologies (Aarnet and Savvis). The proposed approach ensures rapid detection and near-real-time mitigation, with response times ranging from 0 to 10 seconds as the respective controllers are programmed to advertise the flows by each switch every 10 seconds. This information/flow is further processed by the controllers and passed to the IDS via TCP. The IDS, upon receiving this, predicts a binary classification using the XGBoost algorithm and broadcasts it back to all controllers. With this information, the appropriate controller activates its mitigation module and blocks the malicious hosts for a near-real-time duration. Despite of a complicated setup of a multicontroller environment with a central IDS trained using a primary dataset obtained from SDN itself, the performance metrics manages to stand on par with the recent works on SDN DDoS security. Finally, the developed system is able to both detect and perform near real time mitigation of DDoS attacks with an accuracy of 98.5%, precision 97%, and recall 97% respectively.

After the completion of this research work, the following key points are concluded and highlighted:

1. The NMR algorithm for placing multiple (newly added) SDN controllers compared with the Bat algorithm has been discussed. Hence, the issue of CPP/MCPP is solved effectively.

2. An intelligent traffic-driven controller load-balancing strategy using ML for multiple controllers to route the highest priority traffic through the shortest path is addressed. Based on the evaluation of various algorithmic outputs for

the training and validation datasets, and also when execution time is taken into account, the CART is found to be the best algorithm.

3. The SDN security issues of ML-based attack (DDoS) detection in application plane and mitigation in the data plane of the optimised multi-controller topologies are properly discussed. The proposed system performs better results in accuracy 98.5%, precision 97% and recall 97% respectively.

5.2 Recommendations

This study is limited to presenting a novel population-based meta-heuristic method, the NMR algorithm, as an approach for multi-controller placement over an SDN environment based on SC, CC, and load-balancing among the controllers and making a comparison with the Bat algorithm. The NMR algorithm from references shows its superior result, while there are many other meta-heuristics like WOA, ALO and others GA, PSO, SA and GWO, each provides an adaptive trade-off for maintaining the balance in a complex search. Hence, additional comparative analysis with the other above mentioned algorithms are set as future enhancement.

To enhance the proposed system's performance further, there is still room for improvement. Researchers might be able to improve the network by testing the proposed approach on larger and more complex topologies, including real-world scenarios; adding more traffic flows, types and applications; and testing the model for present NetworkX topologies.

In this work, only two key performance metrics – latency and load balancing – are considered based on their priority features designed under the scope of our study, while other general metrics in the Table 2.4 are considered as future enhancements.

Since traditional SL produces acceptable results and advanced ML (for example, DNN, LSTM) requires more data, sophisticated hardware, and increased computational complexity, SL rather than DL is chosen in this study. Therefore, applynig DL could be the further research in the future. Also, we have tried to focus DDoS attack detection and mitigation on optimal multi-controller placement and in a SDN environment.DDoS attacks on the control plane and mitigation on the data plane are discussed. However, we recommend further research on additional SDN-specific architectural challenges and known vulnerabilities to strengthen this work in the future.

Currently, the majority researchers are working on the controller placement problem for data centers. However, they do not offer a particularly efficient approach for large-sized ISP/Telco networks with multi-controller dynamic traffic loads and faults. Thus, CPP during SDN deployment in the ISP/Telco networks is recommended for future research.

5.3 Academic Contributions

Based on the defined scope of works, one review paper and two journal papers were already published, while one more journal paper is under review for possible publication in the prestigious international journal. The abstract of review/journal papers with contributions are summarized in this section.

5.3.1 Paper abstracts and contributions

Title: Controller placement problem during SDN deployment in the ISP/Telco networks: A survey (RP)

Abstract: *With the successful implementation of Software-Defined Networking SDN in data center networking, the way forward for its deployment in the ISP/Telco network is becoming prominent. Small and medium-sized networks may easily adopt SDN. The research on SDN deployment and implementation for a large-scale network is continuing. This paper properly presents the current research status of Controller Placement Problem (CPP) and Multi-CPP (MCPP) over SDN with their specific challenges and provides a comprehensive review of the major performance metrics, i.e., latency, and controller load balancing techniques. This survey highlights the use of network partitioning-based CPP and clustering approaches and their benefits in the context of SDN deployment. Moreover, this paper highlights the importance of implementing SDN and SDN security issues into ISP/Telco networks. Finally, we provide some key areas of ongoing research and discuss the future research direction regarding the various SDN-based Controller Placement (CP) issues in the next-generation IP and advanced networking technologies.*

Contributions:

- provide the current research status of MCPP over SDN with their specific challenges.

- present various facets of network partitions-based CPP and their benefits in the context of SDN deployment.
- focus on two key performance metrics viz. latency and controller load balancing.
- highlight the practical viability or significance of implementing SDN in ISP/Telco networks.
- present security considerations in ISP/Telco networks
- present some of the key areas of ongoing research to give the readers a sense of the current state of knowledge and,
- propose future directions regarding the various CP issues on next-generation IP and advanced networking technologies.

Title: Multi-Controller Placement Optimization Using Naked Mole-Rat Algorithm over Software-Defined Networking Environment (JP1).

Abstract: *SDN is the novel networking paradigm where decoupling of the control plane from the data plane has its inherent advantages. CPP involves placing the optimal number of controllers at the appropriate locations while meeting prerequisites such as latency, load balancing, energy and computational time. To achieve scalability, deployment of multiple controllers on large scale SDN is one of the key challenges. CPP can be addressed as a multi-objective combinatorial optimization problem whose solution is a trade-off between multiple optimization parameters. In this paper, a novel population-based meta-heuristic algorithm viz. NMR Algorithm has been proposed to optimize the location for controller placement based on SC, CC latency while maintaining load balancing among the controllers. The ideas and mechanisms are illustrated using two publicly available standard topologies viz. Ernet and Savvis. The controller localization approach implemented with NMR algorithm has slightly a better result as compared with the Bat algorithm.*

Contributions:

- The NMR algorithm for optimum controller placement is proposed.

- The proposed NMR algorithm is evaluated with different latency values, considering optimum load balancing, and compared with the Bat algorithm.

- Three aspects—(i) the effect on average SC latency; (ii) the effect on global latency; and (iii) the effect of algorithm execution complexity with the increase in controllers—are included in the analysis to find the optimal solution.

Title: Traffic-Driven Controller-Load-Balancing over Multi-Controller Software-Defined Networking Environment (JP2)

Abstract: *Currently, more studies are focusing on traffic classification in software-defined networks (SDNs). Accurate classification and selecting the appropriate controller have benefited from the application of machine learning (ML) in practice. In this research, we study different classification models to see which one best classifies the generated dataset and goes on to be implemented for real-time classification. In our case, the classification and regression tree (CART) classifier produces the best classification results for the generated dataset, and logistic regression is also considerable. Based on the evaluation of various algorithmic outputs for the training and validation datasets, and also when execution time is taken into account, the CART is found to be the best algorithm. While testing the impact of load balancing in a multi-controller SDN environment, in different load case scenarios, we observe network performance parameters like bit rate, packet rate, and jitter. Here, the use of traffic classification-based load balancing improves the bit rate as well as the packet rate of traffic flow on a network and thus considerably enhances throughput. Finally, the reduction in jitter while increasing the controllers confirms the improvement in QoS in a balanced multi-controller SDN environment.*

Contributions:

- A trained model was developed for classification of traffic into video, voice and data category.

- A priority based load balancing algorithm is implemented along with classification algorithm to route highest priority traffic through shortest path improving efficiency of network.

Title: Machine Learning-based Attack Detection and Mitigation with Multi-Controller Placement Optimization over a Software-Defined Networking Environment (JP3)

Abstract: *The increasing complexity and scale of modern software-defined networking demands advanced solutions to address security challenges, particularly distributed denial-of-service (DDoS) attacks in multi-controller environments. Traditional single controller implementations are struggling to effectively counter sophisticated cyber threats, necessitating a faster and scalable solution. This study introduces a novel approach for attack detection and mitigation with optimized multi-controller software-defined networking (SDN) using machine learning (ML). The study focuses on the design, implementation, and assessment of optimal placement of multi-controllers using K-means++ and OPTICS in real topologies and an intrusion detection system (IDS) using the XGBoost classification algorithm to detect and mitigate attacks efficiently with accuracy, precision, and recall of 98.5%,97.0%, and 97.0% respectively. Additionally, the IDS decouples from the controllers, preserves controller resources, and allows for efficient near-real-time attack detection and mitigation. The proposed solution outperforms well by autonomously identifying anomalous behaviors in network by successfully combining controller placement problem (CPP) and DDoS security.*

Contributions:

- Deployment of K-means++ and alternate density-based OPTICS algorithms to determine the optimal number and optimal placement of multi-controllers.
- Development of novel algorithms for multi-controller SDN DDoS threat detection and resolution.
- Implementation of DDoS attack detection mechanism in application plane and mitigation mechanism in data plane.

Title: Ddos Attack Detection and Mitigation Using Machine Learning in a Balanced Multi-Controller Software-Defined Networking (CP1)

Abstract:

This research work proposes an efficient approach for establishing a well-balanced multicontroller software-defined networking (SDN) system capable of detecting and

mitigating distributed denial of service (DDoS) attacks using various machine learning (ML) models. Due to the limited capacity of a single controller, handling the high traffic generated by numerous intelligent devices becomes challenging. To address this, multiple controllers are utilized. However, dynamic network traffic can lead to an uneven distribution of loads among controllers, necessitating a balanced network. To achieve this, switch migration based on latency is employed to ensure proper load balancing. Under normal circumstances, this maintains network stability, but the occurrence of a DDoS attack disrupts this equilibrium. The proposed approach continuously monitors incoming packets and identifies DDoS attacks using ML models such as XGBoost, LightGBM, and CatBoost. A comparative analysis based on test accuracy, precision, and F1 score reveals that CatBoost outperforms XGBoost and LightGBM. The methodology is assessed through emulation in Mininet, with detection and mitigation mechanisms implemented in the RYU controller. Experimental findings indicate enhancements in key performance metrics, including attack detection time, response delay for legitimate requests during an attack, and overall CPU utilization. The system effectively detects and mitigates DDoS attacks within the SDN control plane, thereby improving network security and stability.

5.3.2 Summary of Contributions

The major contributions of presented research are:

- The current research status of MCPP over SDN and their specific challenges are provided; various facets of network partition-based CPP and their benefits in the context of SDN deployment are presented; and two key performance metrics, viz. latency and controller load balance, are focused, and the practical viability or significance of implementing SDN in ISP/Telco networks are highlighted. (*RP-Published)

- A novel population-based meta-heuristic method, NMR algorithm, is proposed to position controllers on the site more effectively based on SC, CC, and load-balancing among the controllers. Two widely accessible standard topologies, Ernet and Savvis, are used to demonstrate the concepts and methods. When compared to the Bat method, the NMR algorithm's controller localization approach yields somewhat superior results. (*JP1-Published)

- Traffic classification-based load balancing in SDN to enhance the system using ML is presented. (*JP2-Published)

- An optimized multi-controller placement and efficient ML-based DDoS attack detection and mitigation system in SDN are focused on. (*JP3-Published)

- DDoS Attack Detection and Mitigation Using Machine Learning in a Balanced Multi-Controller Software-Defined Networking is discussed. (*CP1-Published)

- Reinforcement Learning-Based Multi-Controller Load Balancing in Software-Defined Networking (*JP4-under review)

References

[1] B. R. Dawadi, D. B. Rawat, S. R. Joshi, P. Manzoni, M. M. Keitsch, Migration cost optimization for service provider legacy network migration to software-defined ipv6 network, International Journal of Network Management 31 (4) (2021) e2145.

[2] A. H. Alhilali, A. Montazerolghaem, Artificial intelligence based load balancing in sdn: A comprehensive survey, Internet of Things (2023) 100814.

[3] K. A. Rasol, J. Domingo-Pascual, Multi-level hierarchical controller placement in software defined networking, in: International Networking Conference, Springer, 2021, pp. 131–145.

[4] B. P. R. Killi, S. V. Rao, Controller placement in software defined networks: A comprehensive survey, Computer Networks 163 (2019) 106883.

[5] A. Kumari, A. S. Sairam, Controller placement problem in software-defined networking: A survey, Networks 78 (2) (2021) 195–223.

[6] S. Kaur, J. Singh, Implementation of server load balancing in software defined networking, in: Information Systems Design and Intelligent Applications: Proceedings of Third International Conference INDIA 2016, Volume 2, Springer, 2016, pp. 147–157.

[7] M. Karakus, A. Durresi, Quality of service (qos) in software defined networking (sdn): A survey, Journal of Network and Computer Applications 80 (2017) 200–218.

[8] D. B. Rawat, S. R. Reddy, Software defined networking architecture, security and energy efficiency: A survey, IEEE Communications Surveys & Tutorials 19 (1) (2016) 325–346.

[9] P. Kumar, A. Baliyan, K. R. Prasad, N. Sreekanth, P. Jawarkar, V. Roy, E. T.

Amoatey, et al., Machine learning enabled techniques for protecting wireless sensor networks by estimating attack prevalence and device deployment strategy for 5g networks, Wireless Communications and Mobile Computing 2022 (2022).

[10] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, Y. Liu, A survey of machine learning techniques applied to software defined networking SDN: Research issues and challenges, IEEE Communications Surveys & Tutorials 21 (1) (2018) 393–430.

[11] B. R. Dawadi, D. B. Rawat, S. R. Joshi, Software defined ipv6 network: A new paradigm for future networking., Journal of the Institute of Engineering 15 (2) (2019).

[12] A. Shirmarz, A. Ghaffari, Taxonomy of controller placement problem (cpp) optimization in software defined network SDN: a survey, Journal of Ambient Intelligence and Humanized Computing 12 (12) (2021) 10473–10498.

[13] D. He, J. Chen, X. Qiu, A density algorithm for controller placement problem in software defined wide area networks, The Journal of Supercomputing 79 (5) (2023) 5374–5402.

[14] A. Naseri, M. Ahmadi, L. PourKarimi, Placement of SDN controllers based on network setup cost and latency of control packets, Computer Communications 208 (2023) 15–28.

[15] A. Dvir, Y. Haddad, A. Zilberman, The controller placement problem for wireless SDN, Wireless Networks 25 (2019) 4963–4978.

[16] Y. Fan, L. Wang, X. Yuan, Controller placements for latency minimization of both primary and backup paths in sdns, Computer Communications 163 (2020) 35–50.

[17] A. K. Singh, S. Srivastava, A survey and classification of controller placement problem in SDN, International Journal of Network Management 28 (3) (2018) e2018.

[18] K. Sood, Y. Xiang, The controller placement problem or the controller selection problem?, Journal of communications and information networks 2 (2017) 1–9.

[19] J. Ali, B.-h. Roh, An effective approach for controller placement in software-defined internet-of-things (sd-iot), Sensors 22 (8) (2022) 2992.

[20] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, ACM SIGCOMM Computer Communication Review 42 (4) (2012) 473–478.

[21] N. Firouz, M. Masdari, A. B. Sangar, K. Majidzadeh, A novel controller placement algorithm based on network portioning concept and a hybrid discrete optimization algorithm for multi-controller software-defined networks, Cluster Computing 24 (3) (2021) 2511–2544.

[22] R. Salgotra, U. Singh, The naked mole-rat algorithm, Neural Computing and Applications 31 (12) (2019) 8837–8857.

[23] W. Lin, L. Zhang, The load balancing research of SDN based on ant colony algorithm with job classification, in: 2016 2nd Workshop on Advanced Research and Technology in Industry Applications (WARTIA-16), Atlantis Press, 2016, pp. 472–476.

[24] B. Sapkota, B. R. Dawadi, S. R. Joshi, Controller placement problem during SDN deployment in the isp/telco networks: A survey, Engineering Reports 6 (2) (2024) e12801.

[25] H. Gasmelseed, R. Ramar, Traffic pattern–based load-balancing algorithm in software-defined network using distributed controllers, International Journal of Communication Systems 32 (17) (2019) e3841.

[26] A. Gupta, A. Raj, M. Arora, et al., Ip traffic classification of 4g network using machine learning techniques, in: 2021 5th International conference on computing methodologies and communication (ICCMC), IEEE, 2021, pp. 127–132.

[27] S. Dong, Multi class svm algorithm with active learning for network traffic classification, Expert Systems with Applications 176 (2021) 114885.

[28] B. Isong, R. R. S. Molose, A. M. Abu-Mahfouz, N. Dladlu, Comprehensive review of SDN controller placement strategies, IEEE Access 8 (2020) 170070–170092.

[29] H. Babbar, S. Parthiban, G. Radhakrishnan, S. Rani, A genetic load balancing algorithm to improve the qos metrics for software defined networking for multimedia applications, Multimedia Tools and Applications 81 (7) (2022) 9111–9129.

[30] V. Hnamte, A. A. Najar, H. Nhung-Nguyen, J. Hussain, M. N. Sugali, Ddos attack detection and mitigation using deep neural network in sdn environment, Computers & Security 138 (2024) 103661.

[31] A. A. Seyedkolaei, S. A. H. Seno, A. Moradi, R. Budiarto, Cost-effective survivable controller placement in software-defined networks, IEEE Access 9 (2021) 129130–129140.

[32] I. Dhiab, Y. Barouni, S. Khalfallah, J. Ben Hadj Slama, Performance evaluation of a hybrid ip/sdn network in data centre network architectures, IET Communications 13 (9) (2019) 1185–1191.

[33] H. I. Kobo, A. M. Abu-Mahfouz, G. P. Hancke, Efficient controller placement and reelection mechanism in distributed control system for software defined wireless sensor networks, Transactions on Emerging Telecommunications Technologies 30 (6) (2019) e3588.

[34] T. Das, M. Gurusamy, Controller placement for resilient network state synchronization in multi-controller sdn, IEEE Communications Letters 24 (6) (2020) 1299–1303.

[35] O. Hohlfeld, J. Kempf, M. Reisslein, S. Schmid, N. Shah, Guest editorial scalability issues and solutions for software defined networks, IEEE Journal on selected areas in communications 36 (12) (2018) 2595–2602.

[36] M. Kobayashi, S. Seetharaman, G. Parulkar, G. Appenzeller, J. Little, J. Van Reijendam, P. Weissmann, N. McKeown, Maturing of openflow and software-defined networking through deployments, Computer Networks 61 (2014) 151–175.

[37] R. Wang, Y. Li, M. Xue, B. Zhao, Y. Yin, Y. Li, A survey on security issues of sdn controllers, in: China Conference on Networking, Springer, 2023, pp. 182–206.

[38] N. S. Radam, S. T. F. Al-Janabi, K. S. Jasim, Multi-controllers placement

optimization in SDN by the hybrid hsa-pso algorithm, Computers 11 (7) (2022) 111.

[39] J. A. Rahim, R. Nordin, O. A. Amodu, Open-source software defined networking controllers: State-of-the-art, challenges and solutions for future network providers., Computers, Materials & Continua 80 (1) (2024).

[40] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, S. A. Mehdi, An architectural evaluation of sdn controllers, in: 2013 IEEE international conference on communications (ICC), IEEE, 2013, pp. 3504–3508.

[41] H. Zhong, Q. Lin, J. Cui, R. Shi, L. Liu, et al., An efficient SDN load balancing scheme based on variance analysis for massive mobile users, Mobile Information Systems 2015 (2015).

[42] D. B. Hoang, M. Pham, On software-defined networking and the design of SDN controllers, in: 2015 6th International Conference on the Network of the Future (NOF), IEEE, 2015, pp. 1–3.

[43] T. Hu, Z. Guo, P. Yi, T. Baker, J. Lan, Multi-controller based software-defined networking: A survey, IEEE access 6 (2018) 15980–15996.

[44] O. Salman, I. H. Elhajj, A. Kayssi, A. Chehab, Sdn controllers: A comparative study, in: 2016 18th mediterranean electrotechnical conference (MELECON), IEEE, 2016, pp. 1–6.

[45] S. Ahmad, A. H. Mir, Sdn interfaces: protocols, taxonomy and challenges, International Journal of Wireless and Microwave Technologies 12 (2) (2022) 11–32.

[46] S.-K. Yoon, Z. Khalib, N. Yaakob, A. Amir, Controller placement algorithms in software defined network-a review of trends and challenges, in: MATEC Web of Conferences, Vol. 140, EDP Sciences, 2017, p. 01014.

[47] A. Kumari, A. S. Sairam, A survey of controller placement problem in software defined networks, arXiv preprint arXiv:1905.04649 (2019).

[48] G. Wang, Y. Zhao, J. Huang, W. Wang, The controller placement problem in software defined networking: A survey, IEEE Network 31 (5) (2017) 21–27.

[49] W. Chen, C. Chen, X. Jiang, L. Liu, Multi-controller placement towards SDN based on louvain heuristic algorithm, IEEE Access 6 (2018) 49486–49497.

[50] M. J. Abdel-Rahman, E. A. Mazied, A. MacKenzie, S. Midkiff, M. R. Rizk, M. El-Nainay, On stochastic controller placement in software-defined wireless networks, in: 2017 IEEE Wireless Communications and Networking Conference (WCNC), IEEE, 2017, pp. 1–6.

[51] J. Hollinghurst, A. Ganesh, T. Baugé, Controller placement methods analysis, in: 2016 6th International Conference on Information Communication and Management (ICICM), IEEE, 2016, pp. 239–244.

[52] M. T. I. ul Huque, W. Si, G. Jourjon, V. Gramoli, Large-scale dynamic controller placement, IEEE Transactions on Network and Service Management 14 (1) (2017) 63–76.

[53] A. Ksentini, M. Bagaa, T. Taleb, I. Balasingham, On using bargaining game for optimal placement of SDN controllers, in: 2016 IEEE International Conference on Communications (ICC), IEEE, 2016, pp. 1–6.

[54] B. Zhang, X. Wang, M. Huang, Multi-objective optimization controller placement problem in internet-oriented software defined network, Computer Communications 123 (2018) 24–35.

[55] Y. Zhang, L. Cui, W. Wang, Y. Zhang, A survey on software defined networking with multiple controllers, Journal of Network and Computer Applications 103 (2018) 101–118.

[56] J. Lu, Z. Zhang, T. Hu, P. Yi, J. Lan, A survey of controller placement problem in software-defined networking, IEEE Access 7 (2019) 24290–24307.

[57] T. Das, V. Sridharan, M. Gurusamy, A survey on controller placement in SDN, IEEE communications surveys & tutorials 22 (1) (2019) 472–503.

[58] A. Abdelaziz, A. T. Fong, A. Gani, U. Garba, S. Khan, A. Akhunzada, H. Talebian, K.-K. R. Choo, Distributed controller clustering in software defined networks, PloS one 12 (4) (2017) e0174715.

[59] J. Liu, J. Liu, R. Xie, Reliability-based controller placement algorithm in software defined networking, Computer Science and Information Systems 13 (2)

(2016) 547–560.

[60] P. Xiao, Z.-y. Li, S. Guo, H. Qi, W.-y. Qu, H.-s. Yu, Ak self-adaptive sdn controller placement for wide area networks, Frontiers of Information Technology & Electronic Engineering 17 (7) (2016) 620–633.

[61] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li, T. Li, Density cluster based approach for controller placement problem in large-scale software defined networkings, Computer Networks 112 (2017) 24–35.

[62] H. Kuang, Y. Qiu, R. Li, X. Liu, A hierarchical k-means algorithm for controller placement in sdn-based wan architecture, in: 2018 10th international conference on measuring technology and mechatronics automation (ICMTMA), IEEE, 2018, pp. 263–267.

[63] Z. Fan, J. Yao, X. Yang, Z. Wang, X. Wan, A multi-controller placement strategy based on delay and reliability optimization in SDN, in: 2019 28th wireless and optical communications conference (WOCC), IEEE, 2019, pp. 1–5.

[64] L. F. Müller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspary, M. P. Barcellos, Survivor: An enhanced controller placement strategy for improving sdn survivability, in: 2014 IEEE Global Communications Conference, IEEE, 2014, pp. 1909–1915.

[65] P. Vizarreta, C. M. Machuca, W. Kellerer, Controller placement strategies for a resilient sdn control plane, in: 2016 8th international workshop on resilient networks design and modeling (RNDM), IEEE, 2016, pp. 253–259.

[66] F. Bannour, S. Souihi, A. Mellouk, Scalability and reliability aware sdn controller placement strategies, in: 2017 13th International Conference on Network and Service Management (CNSM), IEEE, 2017, pp. 1–4.

[67] M. Khorramizadeh, V. Ahmadi, Capacity and load-aware software-defined network controller placement in heterogeneous environments, Computer Communications 129 (2018) 226–247.

[68] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid, H. J. Chao, Star: Preventing flow-table overflow in software-defined networks, Computer Networks 125 (2017) 15–25. doi:https://doi.org/10.1016/j.comnet.2017.04.046.

[69] M. Dhar, A. Debnath, B. K. Bhattacharyya, M. K. Debbarma, S. Debbarma, A comprehensive study of different objectives and solutions of controller placement problem in software-defined networks, Transactions on Emerging Telecommunications Technologies 33 (5) (2022) e4440.

[70] H. Selvi, S. Güner, G. Gür, F. Alagöz, The controller placement problem in software defined mobile networks (SDMN), Software defined mobile networks (SDMN): beyond LTE network architecture (2015) 129–147.

[71] K.-Y. Wang, S.-J. Kao, M.-T. Kao, An efficient load adjustment for balancing multiple controllers in reliable SDN systems, in: 2018 IEEE international conference on applied system invention (ICASI), IEEE, 2018, pp. 593–596.

[72] Y. Fan, Y. Xia, W. Liang, X. Zhang, Latency-aware reliable controller placements in SDNs, in: Communications and Networking: 11th EAI International Conference, ChinaCom 2016 Chongqing, China, September 24–26, 2016, Proceedings, Part II 11, Springer, 2018, pp. 152–162.

[73] A. Abuarqoub, A review of the control plane scalability approaches in software defined networking, Future Internet 12 (3) (2020) 49.

[74] A. L. Abraham, B. Hrishikesh, M. T. Atehar, B. N. Hegde, A. Giri, Onos sdn framework: assessing the impact of single and multi-controller architectures on network efficiency, in: International Conference on Smart Computing and Communication, Springer, 2024, pp. 385–396.

[75] M. Karakus, A. Durresi, A survey: Control plane scalability issues and approaches in software-defined networking (SDN), Computer Networks 112 (2017) 279–293.

[76] A. Javadpour, F. Ja'fari, P. Pinto, W. Zhang, Mapping and embedding infrastructure resource management in software defined networks, Cluster Computing 26 (1) (2023) 461–475.

[77] S. H. Yeganeh, A. Tootoonchian, Y. Ganjali, On scalability of software-defined networking, IEEE Communications Magazine 51 (2) (2013) 136–141.

[78] A. A. Neghabi, N. J. Navimipour, M. Hosseinzadeh, A. Rezaee, Load balancing mechanisms in the software defined networks: a systematic and comprehensive review of the literature, IEEE access 6 (2018) 14159–14178.

[79] S. P. D ARCHANA, A study on load balancing in wireless sensor network, Journal homepage: www. ijrpr. com ISSN 2582 7421.

[80] B. Pourghebleh, V. Hayyolalam, A comprehensive and systematic review of the load balancing mechanisms in the internet of things, Cluster Computing 23 (2020) 641–661.

[81] L. Zhu, M. M. Karim, K. Sharif, F. Li, X. Du, M. Guizani, SDN controllers: Benchmarking & performance evaluation, arXiv preprint arXiv:1902.04491 (2019).

[82] A. Javadpour, Providing a way to create balance between reliability and delays in sdn networks by using the appropriate placement of controllers, Wireless Personal Communications 110 (2020) 1057–1071.

[83] G. Wang, Y. Zhao, J. Huang, Y. Wu, An effective approach to controller placement in software defined wide area networks, IEEE Transactions on Network and Service Management 15 (1) (2017) 344–355.

[84] L. Mamushiane, J. Mwangama, A. A. Lysko, Controller placement optimization for software defined wide area networks (sdwan) (2021).

[85] K. A. Rasol, J. Domingo-Pascual, Evaluation of joint controller placement for latency and reliability-aware control plane, in: 2021 Eighth International Conference on Software Defined Systems SDS), IEEE, 2021, pp. 01–07.

[86] B. P. R. Killi, S. V. Rao, Optimal model for failure foresight capacitated controller placement in software-defined networks, IEEE Communications Letters 20 (6) (2016) 1108–1111.

[87] A. K. Singh, S. Maurya, N. Kumar, S. Srivastava, Heuristic approaches for the reliable SDN controller placement problem, Transactions on Emerging Telecommunications Technologies 31 (2) (2020) e3761.

[88] C. Gao, H. Wang, F. Zhu, L. Zhai, S. Yi, A particle swarm optimization algorithm for controller placement problem in software defined network, in: Algorithms and Architectures for Parallel Processing: 15th International Conference, ICA3PP 2015, Zhangjiajie, China, November 18-20, 2015, Proceedings, Part III 15, Springer, 2015, pp. 44–54.

[89] A. K. Singh, N. Kumar, S. Srivastava, Pso and tlbo based reliable placement of controllers in SDN, International Journal of Computer Network and Information Security 11 (2) (2019) 36–42.

[90] C. Wang, H. Ni, L. Liu, Gravcpa: Controller placement algorithm based on traffic gravitation in SDN, Journal of Control Science and Engineering 2022 (2022).

[91] M. Dhar, B. K. Bhattacharyya, M. Kanti Debbarma, S. Debbarma, A new optimization technique to solve the latency aware controller placement problem in software defined networks, Transactions on Emerging Telecommunications Technologies 32 (10) (2021) e4316.

[92] B. R. Dawadi, D. B. Rawat, S. R. Joshi, P. Manzoni, Legacy network integration with sdn-ip implementation towards a multi-domain sodip6 network environment, Electronics 9 (9) (2020) 1454.

[93] C. Veness, Calculate distance and bearing between two latitude/longitude points using haversine formula in javascript, Movable Type Scripts (2011).

[94] G. Wang, Y. Zhao, J. Huang, R. M. Winter, On the data aggregation point placement in smart meter networks, in: 2017 26th International Conference on Computer Communication and Networks (ICCCN), IEEE, 2017, pp. 1–6.

[95] S. Skiena, Implementing discrete mathematics: combinatorics and graph theory with Mathematica, Addison-Wesley Longman Publishing Co., Inc., 1991.

[96] A. Ghosh, M. Manoranjitham, et al., A study on load balancing techniques in SDN, International Journal of Engineering & Technology 7 (174) (2018) 03.

[97] Y. Zhou, K. Zheng, W. Ni, R. P. Liu, Elastic switch migration for control plane load balancing in SDN, IEEE Access 6 (2018) 3909–3919.

[98] M. R. Belgaum, S. Musa, M. M. Alam, M. M. Su'ud, A systematic review of load balancing techniques in software-defined networking, IEEE Access 8 (2020) 98612–98636.

[99] S. K. Askar, Adaptive load balancing scheme for data center networks using software defined network, Science Journal of University of Zakho 4 (2) (2016) 275–286.

[100] S. Ahmad, A. H. Mir, Scalability, consistency, reliability and security in SDN controllers: a survey of diverse SDN controllers, Journal of Network and Systems Management 29 (2021) 1–59.

[101] J. Ali, R. H. Jhaveri, M. Alswailim, B.-h. Roh, Escalb: An effective slave controller allocation-based load balancing scheme for multi-domain SDN-enabled-iot networks, Journal of King Saud University-Computer and Information Sciences 35 (6) (2023) 101566.

[102] P. Kumari, D. Thakur, Load balancing in software defined network, in: International Journal of Computer Sciences and Engineering, Vol. 5, 2017, pp. 227–232.

[103] L. Li, Q. Xu, Load balancing researches in SDN: A survey, in: 2017 7th IEEE International Conference on Electronics Information and Emergency Communication (ICEIEC), IEEE, 2017, pp. 403–408.

[104] VMware, What is load balancing?, accessed: Dec. 5, 2025.
URL https://www.vmware.com/topics/load-balancing

[105] T.-L. Lin, C.-H. Kuo, H.-Y. Chang, W.-K. Chang, Y.-Y. Lin, A parameterized wildcard method based on SDN for server load balancing, in: 2016 international conference on networking and network applications (NaNA), IEEE, 2016, pp. 383–386.

[106] A. Javadpour, A. K. Sangaiah, P. Pinto, F. Ja'fari, W. Zhang, A. M. H. Abadi, H. Ahmadi, An energy-optimized embedded load balancing using dvfs computing in cloud data centers, Computer Communications 197 (2023) 255–266.

[107] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, H. J. Chao, Improving the performance of load balancing in software-defined networks through load variance-based synchronization, Computer Networks 68 (2014) 95–109.

[108] B. R. Killi, S. V. Rao, Poly-stable matching based scalable controller placement with balancing constraints in SDN, Computer Communications 154 (2020) 82–91.

[109] H. Zhong, J. Sheng, Y. Xu, J. Cui, Scplbs: a smart cooperative platform for load balancing and security on SDN distributed controllers, Peer-to-peer

Networking and Applications 12 (2) (2019) 440–451.

[110] Y. Zhou, M. Zhu, L. Xiao, L. Ruan, W. Duan, D. Li, R. Liu, M. Zhu, A load balancing strategy of SDN controller based on distributed decision, in: 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, IEEE, 2014, pp. 851–856.

[111] K. Konglar, Y. Somchit, Load distribution of software-defined networking based on controller performance, in: 2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE), IEEE, 2018, pp. 1–6.

[112] H. Babbar, S. Rani, M. Masud, S. Verma, D. Anand, N. Jhanjhi, Load balancing algorithm for migrating switches in software-defined vehicular networks (2021).

[113] T. Zhang, P. Giaccone, A. Bianco, S. De Domenico, The role of the inter-controller consensus in the placement of distributed SDN controllers, Computer Communications 113 (2017) 1–13.

[114] W. Lan, F. Li, X. Liu, Y. Qiu, A dynamic load balancing mechanism for distributed controllers in software-defined networking, in: 2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), IEEE, 2018, pp. 259–262.

[115] N. Lin, Q. Zhao, L. Zhao, A. Hawbani, L. Liu, G. Min, A novel cost-effective controller placement scheme for software-defined vehicular networks, IEEE Internet of Things Journal 8 (18) (2021) 14080–14093.

[116] K. Yang, D. Guo, B. Zhang, B. Zhao, Multi-controller placement for load balancing in SDWAN, IEEE Access 7 (2019) 167278–167289.

[117] A. Al-Darrab, I. Al-Darrab, A. Rushdi, Software-defined networking load distribution technique for an internet service provider, Journal of Network and Computer Applications 155 (2020) 102547.

[118] O. Belkadi, A. Vulpe, Y. Laaziz, S. Halunga, Ml-based traffic classification in an SDN-enabled cloud environment, Electronics 12 (2) (2023) 269.

[119] J. Liu, Q. Xu, Machine learning in software defined network, in: 2019 IEEE

3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), IEEE, 2019, pp. 1114–1120.

[120] K. Sivamayil, E. Rajasekar, B. Aljafari, S. Nikolovski, S. Vairavasundaram, I. Vairavasundaram, A systematic study on reinforcement learning based applications, Energies 16 (3) (2023) 1512.

[121] M. Shafiq, X. Yu, A. A. Laghari, L. Yao, N. K. Karn, F. Abdessamia, Network traffic classification techniques and comparative analysis using machine learning algorithms, in: 2016 2nd IEEE International Conference on Computer and Communications (ICCC), IEEE, 2016, pp. 2451–2455.

[122] Z. A. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, G. Noubir, Application-awareness in SDN, in: Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, 2013, pp. 487–488.

[123] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, H. S. Mamede, Machine learning in software defined networks: Data collection and traffic classification, in: 2016 IEEE 24th International conference on network protocols (ICNP), IEEE, 2016, pp. 1–5.

[124] M. M. Raikar, S. Meena, M. M. Mulla, N. S. Shetti, M. Karanandi, Data traffic classification in software defined networks SDN using supervised-learning, Procedia Computer Science 171 (2020) 2750–2759.

[125] M. Amiri, H. Al Osman, S. Shirmohammadi, Game-aware and SDN-assisted bandwidth allocation for data center networks, in: 2018 IEEE conference on multimedia information processing and retrieval (MIPR), IEEE, 2018, pp. 86–91.

[126] B. Wang, J. Zhang, Z. Zhang, L. Pan, Y. Xiang, D. Xia, Noise-resistant statistical traffic classification, IEEE Transactions on Big Data 5 (4) (2017) 454–466.

[127] W.-J. Eom, Y.-J. Song, C.-H. Park, J.-K. Kim, G.-H. Kim, Y.-Z. Cho, Network traffic classification using ensemble learning in software-defined networks, in: 2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), IEEE, 2021, pp. 089–092.

[128] S. Ejaz, Z. Iqbal, P. A. Shah, B. H. Bukhari, A. Ali, F. Aadil, Traffic load balancing using software defined networking (SDN) controller as virtualized

network function, IEEE Access 7 (2019) 46646–46658.

[129] N. T. Hai, D.-S. Kim, Efficient load balancing for multi-controller in SDN-based mission-critical networks, in: 2016 IEEE 14th International Conference on Industrial Informatics (INDIN), IEEE, 2016, pp. 420–425.

[130] A. K. Mousa, M. N. Abdullah, A survey on load balancing, routing, and congestion in SDN, Engineering and Technology Journal 40 (10) (2022) 1284–1294.

[131] A. Sapkota, B. B. R. Dawadi, C. S. R. Joshi, et al., Multi-controller placement optimization using naked mole-rat algorithm over software-defined networking environment, Journal of Computer Networks and Communications 2022 (2022).

[132] H. Xue, K. T. Kim, H. Y. Youn, Dynamic load balancing of software-defined networking based on genetic-ant colony optimization, Sensors 19 (2) (2019) 311.

[133] A. R. Mohammed, S. A. Mohammed, S. Shirmohammadi, Machine learning and deep learning based traffic classification and prediction in software defined networking, in: 2019 IEEE International Symposium on Measurements & Networking (M&N), IEEE, 2019, pp. 1–6.

[134] A. Malik, R. de Fréin, M. Al-Zeyadi, J. Andreu-Perez, Intelligent SDN traffic classification using deep learning: Deep-SDN, in: 2020 2nd International Conference on Computer Communication and the Internet (ICCCI), IEEE, 2020, pp. 184–189.

[135] B. Babayigit, B. Ulu, Deep learning for load balancing of sdn-based data center networks, International Journal of Communication Systems 34 (7) (2021) e4760.

[136] M. Yang, Y. Li, D. Jin, L. Zeng, X. Wu, A. V. Vasilakos, Software-defined and virtualized future mobile and wireless networks: A survey, Mobile Networks and Applications 20 (1) (2015) 4–18.

[137] H. AlMomin, A. A. Ibrahim, Detection of distributed denial of service attacks through a combination of machine learning algorithms over software defined network environment, in: 2020 International Congress on Human-Computer

Interaction, Optimization and Robotic Applications (HORA), IEEE, 2020, pp. 1–4.

[138] S. Dong, K. Abbas, R. Jain, A survey on distributed denial of service (ddos) attacks in sdn and cloud computing environments, IEEE Access 7 (2019) 80813–80828.

[139] A. K. Tahirou, K. Konate, M. M. Soidridine, Detection and mitigation of ddos attacks in sdn using machine learning (ml), in: 2023 International Conference on Digital Age & Technological Advances for Sustainable Development (ICDATA), IEEE, 2023, pp. 52–59.

[140] M. Hamdan, E. Hassan, A. Abdelaziz, A. Elhigazi, B. Mohammed, S. Khan, A. V. Vasilakos, M. N. Marsono, A comprehensive survey of load balancing techniques in software-defined network, Journal of Network and Computer Applications 174 (2021) 102856.

[141] D. Melkov, S. Paulikas, Security benefits and drawbacks of software-defined networking, in: 2021 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), IEEE, 2021, pp. 1–4.

[142] M. A. Aladaileh, M. Anbar, I. H. Hasbullah, Y.-W. Chong, Y. K. Sanjalawe, Detection techniques of distributed denial of service attacks on software-defined networking controller–a review, IEEE Access 8 (2020) 143985–143995.

[143] J. Singh, S. Behal, Detection and mitigation of ddos attacks in sdn: A comprehensive review, research challenges and future directions, Computer Science Review 37 (2020) 100279.

[144] M. Di Mauro, G. Galatro, G. Fortino, A. Liotta, Supervised feature selection techniques in network intrusion detection: A critical review, Engineering Applications of Artificial Intelligence 101 (2021) 104216.

[145] D. M. Rajan, D. J. Aravindhar, Detection and mitigation of ddos attack in sdn environment using hybrid cnn-lstm, Migration Letters 20 (S13) (2023) 407–419.

[146] J. N. Bakker, B. Ng, W. K. Seah, Can machine learning techniques be effectively used in real networks against ddos attacks?, in: 2018 27th International Conference on Computer Communication and Networks (ICCCN),

IEEE, 2018, pp. 1–6.

[147] O. Rahman, M. A. G. Quraishi, C.-H. Lung, Ddos attacks detection and mitigation in sdn using machine learning, in: 2019 IEEE world congress on services (SERVICES), Vol. 2642, IEEE, 2019, pp. 184–189.

[148] J. Chen, Y.-t. Yang, K.-k. Hu, H.-b. Zheng, Z. Wang, Dad-mcnn: Ddos attack detection via multi-channel cnn, in: Proceedings of the 2019 11th International Conference on Machine Learning and Computing, 2019, pp. 484–488.

[149] V. Morfino, S. Rampone, Towards near-real-time intrusion detection for iot devices using supervised learning and apache spark, Electronics 9 (3) (2020) 444.

[150] M. A. Mohsin, A. H. Hamad, Performance evaluation of sdn ddos attack detection and mitigation based random forest and k-nearest neighbors machine learning algorithms, Revue d'Intelligence Artificielle 36 (2) (2022) 233.

[151] N. Meti, D. Narayan, V. Baligar, Detection of distributed denial of service attacks using machine learning algorithms in software defined networks, in: 2017 international conference on advances in computing, communications and informatics (ICACCI), IEEE, 2017, pp. 1366–1371.

[152] P. Valizadeh, A. Taghinezhad-Niar, A fault tolerant multi-controller framework for sdn ddos attacks detection, International Journal of Web Research 5 (1) (2022) 1–7.

[153] T. G. Gebremeskel, K. A. Gemeda, T. G. Krishna, P. J. Ramulu, Ddos attack detection and classification using hybrid model for multicontroller sdn, Wireless Communications and Mobile Computing 2023 (1) (2023) 9965945.

[154] T. Pandikumar, F. Atkilt, C. A. Hassen, Early detection of ddos attacks in a multi-controller based sdn, International Journal of Engineering Science 13422 (2017).

[155] U. Gurusamy, H. K, M. MSK, Detection and mitigation of udp flooding attack in a multicontroller software defined network using secure flow management model, Concurrency and Computation: Practice and Experience 31 (20) (2019) e5326.

[156] H. Huang, H. Yin, G. Min, H. Jiang, J. Zhang, Y. Wu, Data-driven information plane in software-defined networking, IEEE Communications Magazine 55 (6) (2017) 218–224.

[157] R. Salgotra, U. Singh, G. Singh, N. Mittal, A. H. Gandomi, A self-adaptive hybridized differential evolution naked mole-rat algorithm for engineering optimization problems, Computer Methods in Applied Mechanics and Engineering 383 (2021) 113916.

[158] S. Avallone, S. Guadagno, D. Emma, A. Pescape, G. Ventre, D-itg distributed internet traffic generator, in: First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings., IEEE, 2004, pp. 316–317.

[159] G. Karn, B. Sapkota, B. R. Dawadi, Traffic classification and load balancing in sdn environment (2023).

[160] M. Rostami, S. Goli-Bidgoli, An overview of qos-aware load balancing techniques in SDN-based iot networks, Journal of Cloud Computing 13 (1) (2024) 89.

[161] V. Punitha, C. Mala, Traffic classification for efficient load balancing in server cluster using deep learning technique, The Journal of Supercomputing 77 (8) (2021) 8038–8062.

[162] J. Opitz, S. Burst, Macro f1 and macro f1, arXiv preprint arXiv:1911.03347 (2019).

[163] M. Grandini, E. Bagli, G. Visani, Metrics for multi-class classification: an overview, arXiv preprint arXiv:2008.05756 (2020).

[164] F. Al-Haidari, M. Sqalli, K. Salah, Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources, in: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, Vol. 2, IEEE, 2013, pp. 256–261.

[165] M. Awad, N. Kara, A. Leivadeas, Utilization prediction-based vm consolidation approach, Journal of Parallel and Distributed Computing 170 (2022) 24–38.

[166] D. Arthur, S. Vassilvitskii, et al., k-means++: The advantages of careful seeding, in: Soda, Vol. 7, 2007, pp. 1027–1035.

[167] M. Ankerst, M. M. Breunig, H.-P. Kriegel, J. Sander, Optics: Ordering points to identify the clustering structure, ACM Sigmod record 28 (2) (1999) 49–60.

[168] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise, in: kdd, Vol. 96, 1996, pp. 226–231.

[169] R. J. Campello, P. Kröger, J. Sander, A. Zimek, Density-based clustering, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 10 (2) (2020) e1343.

[170] H.-P. Kriegel, P. Kröger, J. Sander, A. Zimek, Density-based clustering, Wiley interdisciplinary reviews: data mining and knowledge discovery 1 (3) (2011) 231–240.

[171] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo, IEEE Journal on Selected Areas in Communications 29 (9) (2011) 1765–1775.

[172] N. Ahuja, G. Singal, D. Mukhopadhyay, N. Kumar, Automated ddos attack detection in software defined networking, Journal of Network and Computer Applications 187 (2021) 103108.

[173] S. Manickam, A. H. B. AIghuraibawi, R. Abdullah, Z. A. A. Alyasseri, K. H. Abdulkareem, M. A. Mohammed, A. Alani, Labelled dataset on distributed denial-of-service (ddos) attacks based on internet control message protocol version 6 (icmpv6), Wireless Communications and Mobile Computing 2022 (1) (2022) 8060333.

[174] P. Manso, J. Moura, C. Serrão, Sdn-based intrusion detection system for early detection and mitigation of ddos attacks, Information 10 (3) (2019) 106.

[175] N. Sanghani, G. Vaghela, B. Borisaniya, Detecting distributed denial of service (ddos) attacks in a multi-controller sdn environment utilizing machine learning, in: International Conference on Computing Science, Communication and Security, Springer, 2024, pp. 117–132.

[176] M. Abdulaziz, B. Al-motairy, M. Al-ghamdi, N. Al-qahtani, Building a personalized fitness recommendation application based on sequential information, International Journal of Advanced Computer Science and Applications 12 (1) (2021).

[177] U. B. Clinton, N. Hoque, K. Robindro Singh, Classification of ddos attack traffic on sdn network environment using deep learning, Cybersecurity 7 (1) (2024) 23.

[178] Y. Al-Dunainawi, B. R. Al-Kaseem, H. S. Al-Raweshidy, Optimized artificial intelligence model for ddos detection in sdn environment, IEEE Access (2023).

[179] A. Hirsi, L. Audah, A. Salh, M. A. Alhartomi, S. Ahmed, Detecting ddos threats using supervised machine learning for traffic classification in software defined networking, IEEE Access (2024).