An SDN-Based Approach Using RYU Controller for Load Balancing and Performance Evaluation in Hybrid Networks with Machine Learning Algorithms

by

Chaity Rani Ghosh
21101191
Niloy Ahsan
21101255
Mahfujul Islam
21101070
Rady Noor
21101038
Abid MAshrafi
21101075

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
February 2025

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing the degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Chaity Rani Ghosh

21101191

Niloy Ahsan

21101255

Mahfujul Islam

21101070

Rady Noor

21101038

Abid Mashrafi

21101075

Approval

The thesis titled "An SDN-Based Approach Using RYU Controller for Load Balancing and Performance Evaluation in Hybrid Networks with Machine Learning Algorithms" submitted by

1. Chaity Rani Ghosh (21101191)
2. Niloy Ahsan (21101255)
3. Mahfujul Islam (21101070)
4. Rady Noor (21101038)
5. Abid Mashrafi (21101075)

Of Fall 2024 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on February 3, 2025.

Examining Committee:

Supervisor:
(Member)

Dr. Jannatun Noor Mukta

Associate Professor
Department of Computer Science and Engineering
United International University

Program Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam

Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Dr. Sadia Hamid Kazi

Chairperson and Associate Professor
Department of Computer Science and Engineering
BRAC University

Abstract

Software-defined networking (SDN) is a technology that is transforming network efficiency, particularly in terms of balancing loads [41]. In this work, an analysis of an SDN-based load balancing and performance analysis via machine learning approaches in the RYU controller is presented. In a simulation conducted in a thorough manner in Mininet, performance analysis of SDN in managing network traffic in a range of topologies including tree, star, linear and cluster networks is discussed. In our work, an analysis of the performance impact of SDN load balancing in terms of performance factors including throughput, latency, jitter, and packet loss is discussed. Heavy traffic as example media, voice, VoIP, etc was used for evaluating the performance. We observed a performance improvement via SDN when accompanied by smart techniques in balancing loads is noticed to have a significant impact in terms of dynamically distributing loads and minimizing congestion in networks using our load balancer which is based on Round Robin Scheduler. With such observations, SDN proves to be an effective and efficient mechanism for high-performance and high-scalability networks in current infrastructure requirements.

Keywords: Software-defined Network, Load Balancing, QoS, RYU, Mininet, OpenFlow Protocol, ARP Protocol iperf, D-ITG.

Acknowledgement

Firstly, all praise to the Great Almighty for whom our thesis has been completed without any major interruption.

Secondly, to our supervisor Dr. Jannatun Noor Mukta ma'am for her kind support and advice in our work. She helped us whenever we needed help.

Thirdly, we would like to thank all the faculty members and stuffs for providing us with such a wonderful study environment where we could develop ourselves as well as this research properly

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

ch	Cluster Host
cs	Cluster Switch
LB	Load Balancing
lh	Linear Host
ls	Linear Switch
ML	Machine Learning
$NoLB$	No Load Balancing
QoS	Quality of Service
SDN	Software Defined Network
sh	Star Host
ss	Star Switch
th	Tree Host
ts	Tree Switch

Chapter 1

Introduction

1.1 Introduction

Cloud computing has grown very rapidly, and the traffic caused by the network is complex and needs to be managed better. Real-world distributed cloud environments have to handle a large number of workloads (from real-time to large-scale data transfer), and existing networking is not capable of delivering the performance and scalability needed. Ensuring uninterrupted connectivity, reducing congestion, and improving security in the modern cloud infrastructure today is equally important and has become a critical topic to optimize traffic flow and balance out the network loads[22]. The ever-evolving digital landscape demands an increment of efficiencies, scalability, and security in network infrastructure. Traditional approaches in networking are rigidly architectural and limited by scalability hence they fail to address the growing complexity of modern network environments [67]. More importantly, this challenge is further driven by the widespread adoption of cloud computing, the proliferation of IoT devices, and the rapid drive toward high-bandwidth applications. In response to these pressures, Software-Defined Networking (SDN) has emerged as a solution [55] offering a dynamic, programmable, and extensible method for managing network resources, well-suited to the demands of contemporary and future network environments [56].

In Software-Defined Networking (SDN), the control plane is decoupled from the data plane for better controllability, flexibility, security, and manageability of network resources, operation, and management. The fundamental architecture of SDN is inherently centralized even the routers, switches, firewalls, etc. are controlled by a centralized SDN controller. It is logically divided into three layers: the data plane, the control plane, and the application plane, as depicted in Figure 1.1. Applications and services that define the desired behavior of the network reside in the application plane. The decisions shall be made by the control plane, and it shall manage traffic, while the actual job of moving data across the network shall be done by the data plane [5]. All these layers are put together to create a unified system that enables seamless integration, effective data flow management, and superior security [52].

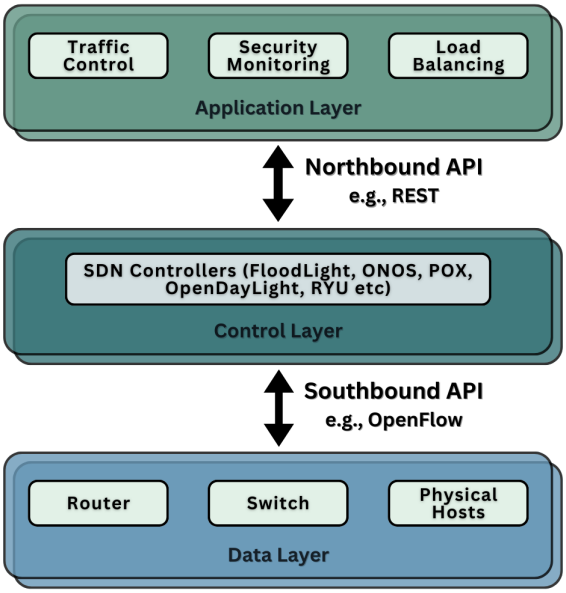


Figure 1.1: Basic SDN Architecture

Efficiency and scalability in network management have recently been in high demand, owing to rapid growth both in Internet traffic and in complex structures of networks. Traditional networking is often inflexible to satisfy these increasing demands and cumbersome to manage [51]. Therefore, SDN has the effect of ensuring a particularly rewarding solution through the decoupling between the control plane and data plane hence offering centralized management and dynamic configuration of network devices. SDN sits in the middle or between the user interface and the underlying hardware, giving it full control through abstraction over the management of the flow of data on top of the network [24] [59].

In this research, we consider SDN because it is centrally managed. Therefore, network administration and configuration become very optimal in order to implement. Another reason is that SDN provides enhanced security due to fine-grained network traffic control and load balancing across nodes [10]. All standardized and managed functionalities through the OpenFlow protocol are in direct contact with network devices and therefore ease network operations.

An important point of this research is to examine the behavior of different types of network traffic especially the distinction between "mice flows" and "elephant flows". Mice flows refer to small and brief data transfer HTTP requests or ping requests which are frequent in cloud environments where user interactions trigger short-lived requests. Mice flow involves minimal data flow but they are highly sensitive to latency. Also, the delays in processing them can have a noticeable impact on user experience, especially in real-time applications. On the other hand, elephant flows represent the transfer of large data volumes such as media traffic like voice, video, or audio streaming or massive file transfers. These kinds of flows need significant bandwidth and depend on high throughput and low latency for smooth and seamless transmission. If elephant flows are not optimized correctly it can cause network congestion, adversely affecting latency-sensitive mice flows [44]. In a custom SDN

topology, these kinds of flows can have a greater impact on the overall performance of the system. If an SDN topology were designed well it can manage both mice and elephant flows intelligently. Here proper resource allocation and prioritization are necessary to ensure proper time-sensitive communications for large data transfers. To optimize load balancing for a variety of traffic patterns in cloud networks, this research attempts to provide information on how these various flow types interact and how SDN may be utilized to manage them efficiently[23].

Using simulations of traffic patterns and network topologies, the study seeks to advance the development of efficient, scalable, and secure network infrastructures that meet the evolving demands of today's digital landscape. By enhancing SDN-based solutions such as RYU, this research aims to demonstrate how dynamic load balancing and traffic management strategies can improve cloud network performance and ensure optimal resource use and enhanced Quality of Service across a variety of applications[43]. Load balancing prevents congestion and maintains continuous flow by spreading the traffic across a large number of routes or devices ensuring that network resources are used efficiently. Static load balancing techniques are often used in traditional networks as a result they can not be flexible and unable to adapt to the new network conditions but for SDN, programmable load balancing is possible which helps to provide real-time traffic modifications according to the state of the network. This helps to reduce the latency, improve the throughput, minimize jitters, and prevent packet loss. As a result, SDN becomes more useful for handling diverse traffic patterns from low-latency services such as Voice over IP (VoIP) to bandwidth-heavy applications for example video streaming [36].

For these kinds of implementations, SDN controllers are introduced. The discussed potentials and benefits can be achieved using SDN controllers like POX, RYU, OpenDaylight, ONOS, os-ken, etc. Many of them are open-source solutions that can be modified, scaled, or adapted to meet certain types of workflow or environments. This kind of flexibility to adapt specific scenarios or needs depending on the network type or environment makes these SDN controllers an effective tool for achieving flexibility. As a result, we have used the RYU controller here [35]. For managing modern cloud computing networks, RYU as a Software-Defined Networking (SDN) controller played an important role here. [38].

RYU is a Python-based API controller that provides flexibility and customizable options for developers and also allows them to create customized network management applications according to their requirements. Recent work in [45] shows the effectiveness of the RYU controller functions in dynamically tracking and storing information such as bandwidth, packet counts, throughput, and round-trip time. Also, their results show us the significance of the RYU controller by providing valuable pieces of information about traffic patterns and network performance [41]. Another study focused on the scalability of the RYU controller in SDN and highlighted the capability of RYU to handle dynamic network environments efficiently. Also, RYU provides us with various troubleshooting options that are important for the scalability and security of cloud networks [20].

1.2 Problem Statement

Traditional network architectures often rely on static load-balancing methods that do not adapt to changing traffic conditions. As a result, they can experience issues such as uneven resource utilization, increased latency, and suboptimal throughput. These limitations are especially pronounced when dealing with different traffic types such as TCP and UDP which have distinct requirements and behaviors. TCP is connection-oriented and sensitive to latency and throughput, while UDP is connectionless and prone to variations in jitter and packet loss.

In SDN, OpenFlow enables the separation of the control plane from the data plane in network devices. This centralized control allows for easier management and configuration of the network through a single interface or controller and also provides an opportunity to implement adaptive load-balancing strategies. Besides, it can scale effectively to accommodate larger and more complex networks by offering a centralized view and control over the network.

However, it remains a challenge to design a load-balancing solution that can dynamically adjust to fluctuating network conditions and handle both TCP and UDP traffic effectively. To ensure optimal network performance it is crucial to evaluate key metrics such as throughput, latency, RTT, and jitter under various traffic loads and conditions.

1.3 Research Objectives

The main goal of this research is to implement and evaluate an SDN-based load-balancing solution that can adaptively manage both TCP and UDP traffic. The specific objectives are as follows:

1. **Design a versatile network topology within an SDN framework** that incorporates tree, star, linear, and cluster configurations to simulate different real-world scenarios.

2. **Develop a load balancing mechanism based on the Round Robin algorithm**, deployed on an SDN controller to distribute traffic dynamically across multiple paths using time domain.

3. **Measure and analyze key performance metrics** including throughput, latency, RTT, jitter, and packet loss for both TCP and UDP traffic. This will help in understanding the impact of load balancing on different types of traffic and identify areas for improvement.

4. **Compare the performance of the SDN-based load balancing system with traditional approaches** by examining TCP and UDP traffic behavior. The comparison will focus on determining how effectively the SDN-based solution can enhance throughput, reduce latency, minimize RTT, and control jitter across diverse network conditions.

Chapter 2

Literature Review

Yzzogh et al. in [55] investigated the application of Software-Defined Networking to cloud computing regarding load optimization. It dynamically distributes the traffic, improving overall resource utilization, system performance, and user experience. Being a scalable and elastic infrastructure for the cloud, it minimizes the chance of any failure. Some new techniques, including fuzzy logic and the ant colony algorithm, also seem promising in terms of enhancement of load distribution with reduced latency. However, implementation complexities and security risks remain a challenge. Most of these strategies remain untested in real-world environments, and the paper underlines the underlying potential of SDN to make a major change in load balancing. The research theme shall be taken forward with the integration of machine learning for optimization.

Vani et al. in [54] improved dynamic load-balancing schemes for Software Defined Networks (SDN) focusing on response time as an important quality of service (QoS) factor that had often been overlooked in previous studies. The authors mentioned the limitations of single-objective optimization methods and proposed a multi-criteria decision-making (MCDM) approach with several competing objectives which offers a more comprehensive solution. This emphasized the shift from static to dynamic load-balancing, leveraging the SDN programmability to enable automatic adaptation with real-time feedback, improving efficiency in the context of dynamic environments. The proposed solution via a three-module system, which are request classification, server monitoring, and optimized load balancing, is compared using the Weighted Sum Method (WSM) for method comparison with several QoS parameters. Experimental results exhibit up to 58% performance improvement, surpassing traditional methods. However, issues of implementation complexity, dependence on accurate server monitoring, and scalability in very large networks remain indicating areas for future study.

Minardi et al. in [61] highlighted the main achievements and challenges of VNE beyond 5G networks and 6G, underlining the necessity for dynamic traffic management. The traditional static traffic models cannot fit into the practical scenario while the introduction of USP aims to meet the growing demands of traffic. The proposed SCA-VNE algorithm optimizes the allocation of resources using real-time data, though its complexity suggests the need for more practical solutions. Future research will be directed at developing scalable and efficient VNE algorithms that

can be adaptable and validated in real-world applications.

Ma et al. in [35] implemented an SDN-based traffic monitoring system using the Ryu controller, leveraging Mininet for network topology simulation and Docker for virtualization. The Ryu controller collected flow information via the OpenFlow protocol enabling real-time monitoring of link load and remaining bandwidth. The methodology involved customizing a data center network topology, deploying monitoring functions, and executing traffic simulations with iperf. Results demonstrated that the system effectively captured traffic states, allowing for improved network management and congestion control. However, challenges included environment configuration complexity and potential conflicts when deploying multiple applications in Docker.

Liu et.al. investigated in [34] about the Software Defined Networking (SDN) approach as a major tool to overcome the network management scale, complexity, and performance, which is discussed in the literature review. The talk discusses how SDN can overcome the limits of traditional networks by providing robust and secure networks in IoT and smart cities. The purpose of SDN is to enable dynamic, efficient traffic routing, congestion control, and fault tolerance through the centralization of network control and programmability. A number of different algorithms, such as Bird Migration and Particle Swarm Optimization have been studied for tasks like multipath routing and load balancing. By optimizing path selection and increasing fault tolerance and network load, these algorithms address scalability and reliability issues. The review also emphasizes the need for further research on SDN in dynamic environments as well as for integration with new technologies.

Omer et al. in [23] conducted a study to contrast load balancing mechanisms for cloud computing with special reference to Software Defined Networking (SDN). A round-robin method had been utilized for the reasonable distribution of requests between servers and the significance of load balancing in regulating traffic control has been recognized. A variety of infrastructure topologies, fat-tree structures included, had been investigated and experimented with in Mininet. Performance metrics such as jitter, throughput, and latency were compared based on SDN controllers ONOS and ODL. The findings indicated that while Apache servers had processed over six million requests, the absence of load balancing had resulted in request drops under heavy traffic. The deployment of HAproxy was shown to achieve significant improvement of response time and prevention of request loss. The study had primarily been founded on HTTP-based load balancing, and recommendations had been made for future studies on protocols such as ICMP and UDP. The study also identified a limitation in low-traffic cases where load balancing had not been required. In the end, the study demonstrated the feasibility of SDN and HAproxy in optimizing cloud server availability while indicating areas for additional research.

Wassie et al. in [69] discussed QoS optimization in SDN by using deep learning models to predict elephant flow and improve network performance. The Random Forest (RF) model employed in predicting the elephant flows was 100% accurate and performed better than the DNN and CNN models which were 99.98% and 98.23% accurate, respectively. The RF model also exhibited a fast prediction time of 9

seconds. Ryu controller and Mininet emulator were used to deploy the models and a confusion matrix was used to assess their performance. Comparative analysis of algorithms like Naive Bayes, DNN, CNN, and RF was given. Some limitations noted are no mention of overfitting, the requirement of training data, and the generalizability of results to other network setups.

Alotaibi et al. in [16] demonstrated that SDN enhances the homogeneity networks' performance by minimizing the handover delay as well as the load balancing, particularly, in heterogenous networks referring to the mix of 4G and 5G, as well as Wi-Fi. Even as SDN propounds control on on the network's parameters in central control it has been established to improve latency, throughput, and packet loss but the correlation between load balancing and handover delay has not been well elucidated. Although prior works consider these KPIs in isolation, the relationship between load-balancing methods and the effects on handover-related delays in multi-connection scenarios has not yet been investigated comprehensively. This study therefore seeks to fill this gap by evaluating its relationship to enhancing SDN-based heterogeneous networks.

Abuthahir et al. in [29] proposed an enhanced dynamic load balancing algorithm for cloud computing, which was a combination of a modified Round Robin approach and Honey Bee behavior-based mechanisms. The hybrid algorithm aimed at optimizing resource utilization and minimizing task completion time by allocating tasks based on priority. The proposed technique has been superior to traditional methods, with considerable enhancement in computational time and efficiency, especially in dynamically loaded systems. Simulation results confirmed the efficiency of the algorithm in providing low completion times and improving system performance. Even though the algorithm had shown promise, scalability in large cloud environments and its performance under highly dynamic load situations had been left as directions for future work.

Kofi and Emmanuel in [51] investigated load balancing in SDN addressing both traditional and AI techniques but none of the proposed solutions incorporate solutions for throughput, overhead, and congestion issues. While increasing the throughput, strategies such as Least-Loaded and Round-Robin do not pay attention to QoS values and include response time. However affording resources in networks, weighted load balancing and multi path routing improve load balancing but are complex and congested. The presented HDW algorithm seems quite reasonable by integrating the dynamic routing with the load scheduling but further validation is needed. In general, the existing approaches enhance the performance based on tracking, but they do not pay attention to QoS and scalability in a comprehensive manner.

Ejaz et al. in [7] discussed load balancing in SDN and presented significant enhancements to load balancing scalability and performance by employing the Master Controller Node, BalanceFlow, and Dynamic Adaptive Load Balancing (DALB) algorithms. With support from Network Function Virtualization (NFV) and Virtual SDN (vSDN) controllers, resource control and operation during traffic rushes are enhanced. However, issues including but not limited to resource constraints, lack of updated load information, dependence on the central controller, and deficient energy

efficiency are still problems that require further research towards the improvement of SDN load balancing.

Liu et al. in [10] discussed a number of open issues of the SDN-based cloud computing paradigm most notably data redundancy and delay in service response. The use of smart devices escalates the amount of data, which is becoming quite redundant in the way it consumes energy, and techniques such as sparse principal component analysis, and matrix completion methods seek to eliminate such data. To reduce measures of service disruption, frameworks that include, but are not limited to, Rahouti's urgency-based categorization and Huang's packet-forwarding approaches are mentioned. The survey also focuses on the integration of SDN with edge-cloud computing to improve QoS, solutions such as multilevel data clustering, and heuristic methods for the assignments of resources. In conclusion, the paper also stresses the importance of the novelties in these cyclical problems.

Katangur et al. in [33] emphasized the necessity of effective resource management due to the increasing number of users. It examined several algorithms including Round Robin (RR), Weighted Round Robin (WRR), and Dynamic Weighted Round Robin (DWRR), and proposed the Priority Weighted Round Robin (PWRR) algorithm as an enhancement over WRR. PWRR incorporated task priority into the scheduling process, enabling more efficient execution based on urgency. The results demonstrated that PWRR outperformed RR and WRR in execution cost, waiting time, turnaround time, and average execution time. It also exhibited significant improvements in high-priority task execution compared to DWRR. Performance analysis was conducted using the CloudSim simulation tool, with findings confirming that PWRR reduced the completion time for high-priority tasks relative to other algorithms. However, the study did not address the scalability of PWRR in large-scale cloud environments, and dynamic load balancing in real-time scenarios remained an unresolved challenge.

Chahlaoui et al. in [32] mentioned that a number of key algorithms, including Equal Cost Multipath, have been developed to enhance link utilization and reliability by finding multiple paths to a destination. Enhancements such as the weighted ECMP and W-ECMP algorithm have proved better performance when weights are applied according to link utilization during heavy loads. The DAMR algorithm will mainly focus on throughput optimization concerning reducing packet loss and packet jitter. Hybrid approaches also exist, effective for load balancing when the networks run both SDN and legacy routing protocols. A hybrid Dijkstra-Repeat algorithm is one example in this class of mechanisms. Besides that, multiple admission control schemes incorporated with multipath routing showed tremendous improvements related to throughput and latency, hence proving the evolution and efficiency of such techniques and satisfying modern network demands.

Praveen et al. in [36] investigated adaptive load balancing techniques for multi-SDN controllers revealing that important improvements have been made in improving the performance of the networks. Several of the algorithms perform both data link and server balancing to lower latency and boost response rates in limited SDN environments. Hybrid routing algorithms enable optimal dynamic server load distribution

techniques, while centralized controllers solve the data plane congestion problem. Multi-path flow scheduling thus increases ductility and load balance techniques illustrated by IP Hash and Least Connection exhibit different performance rates with IP Hash performing better than traditional techniques. Moreover, heuristic approaches have been evolved to avoid switch migrations when there are a large amount of switches between clusters and the dynamic load management approaches prevent the high utilization of a particular link by prioritizing the flows. In summary, the review signifies a marked trend toward dynamic adaptive approaches in load balancing and serves as a strong platform for future research on SDN.

Ye et al. in [70] stated that the ILBPS algorithm combines load-balancing with adaptive heuristics to determine the path of traffic in the SDN more effectively. It uses SSA for traffic decomposition, GCBAC deep learning model for payload prediction, and CNN for noise data. They then employ the artificial bee colony (ABC) algorithm for path determination. Experiments conducted on the GÉANT network using Mininet and Ryu controllers reveal that ILBPS provide better throughput, jitter, and load balancing than existing approaches such as CNN+Load Balancing and ALBLP. However, some issues arising from the system implementation include the computational aspect, and scalability issues relevant to real-time applications that have not been done yet has been conducted on small networks.

Wang et al. in [37] discussed mechatronics also included the improvement in measurement techniques and automation processes for high efficiency and accuracy. Foundational studies have focused on sensor technology, control systems, and different methods of data acquisition, emphasizing the integration of advanced measurement technologies to optimize performance and reliability. Other current trends include the application of AI and machine learning in mechatronics for real-time data processing and decision-making. Challenges identified include limited applicability of certain measurement technologies across diverse environments, over-reliance on theoretical models that may or may not correctly represent real-world complexities, and very few studies have tackled the long-term reliability issues of advanced systems. Some of the future research directions should be to develop adaptable measurement technologies, empirical validation of theoretical models, and stimulate more interdisciplinary collaboration in innovating and enhancing mechatronic systems.

Avinash et al. in [40] presented a fresh approach to the problem of load balancing in the environment of software-defined networking by combining static and dynamic methods for the optimization of resource utilization. The algorithm runs on multiple paths by intelligently distributing traffic using hierarchical multi-controller establishment to avoid congestion and improve performance. It demonstrated a real enhancement in terms of throughput, jitter, and latency on Tree and Dumbbell topologies compared to any other conventional method. Such an evaluation proved the reliability and precision of the algorithm in managing network resources. This hybrid approach will consider other major challenges such as intradomain and inter-domain cost of communication with adaptability to diverse topologies. Many realistic applications are possible such as smart railway management, intrusion detection systems, etc., thereby benefiting the system's efficiency and opening new

avenues of research in SDN-based load balancing.

Cabarkapa et al. in [18] highlighted the development and efficiency of the SDN controllers, with a special concentration on the Ryu and POX controllers. It outlines the evolution of the controllers such as from NOX to POX and lays a comparison of the number of controllers such as Ryu, POX, Trema, Floodlight, and OpenDayLight in terms of parameters for instance the latency, the throughput or even packet loss. Ryu has less latency than other modules in simple network topology while POX is more efficient in handling larger data in complex topology. As seen in many papers, Mininet is used for simulation and the review stresses the requirement for more investigations regarding scalability and security issues in real-life SDN applications.

Arahunashi et al. in [5] assessed different SDN controllers, such as Ryu, Floodlight, ONOS, and OpenDayLight, with a specific focus on performance measurements like latency and throughput using the Cbench tool. ONOS and OpenDayLight are known for their wide range of features, whereas Floodlight provides less and Ryu is suggested for smaller deployments. Controllers were assessed in various network structures—linear, tree, and mesh—using Mininet, with a delay determined through ping and throughput through iperf with different connection loads. The findings show that Floodlight consistently has less delay, while Ryu and ONOS have comparable throughput, but performance decreases as network complexity increases. Future research should investigate more QoS parameters and evaluate a wider range of controllers, taking into account factors such as security and the effects of malicious traffic on performance, as the study's emphasis on delay and throughput and its limited topologies present a need for expansion.

Girish et al. in [59] identified research gaps in employing the SDN-based cloud computing model, with the two main shortcomings being data redundancy and service response delay. With more smart devices, data is redundant and energy is wasted, solutions such as sparse principal component analysis and matrix completion methods seek to minimize the redundancy. To reduce service interruption, approaches that are Rahouti's categorization of services based on urgency level and Huang's packet forwarding schemes are presented. Integration of SDN with edge-cloud computing to improve QoS is also enumerated by the survey together with solutions such as multilevel data aggregation and heuristic algorithms used in the assignment of resources. In conclusion, the paper underscores the centrality of the need to address these concerns in new ways.

Ťaptík et al. in [66] used SDN controllers to play a critical role in managing network functions in Software-Defined Networking. The major available options are Ryu, Floodlight, and OpenMul, among others, and each has different strengths. Ryu is a Python-based controller well-suited for small networks but limited in scalability, while Floodlight, written in Java, offers more scalability and platform support, thus making it more suitable for larger networks. OpenMul, implemented in C, allows high performance but does not offer ongoing development and a user-friendly interface; therefore, its application is limited. A performance comparison using Mininet shows differences in scalability and usability. The way ahead for research should be feature enhancement and long-term support, integrating SDN controllers with

emerging technologies such as IoT and 5G.

Bhaskaran and Supriya in [56] provided a deep overview of SDN architecture, focused on the key components, findings, and limitations, as well as future directions. It represents the three key planes of SDN: the Application Layer, which hosts numerous network applications; the Control Layer, responsible for the management of the network through controllers communicating with data plane devices via southbound APIs; and the Data Plane, comprising the switches and routers that forward packets following instructions from the control layer. It discusses, with the help of emulators like Mininet and MiniEdit, custom topologies, along with performance metrics in terms of throughput, packet loss, and retransmission rates for IoT applications. The study points out some important limitations, including large infrastructure changes that would be needed for SDN implementation, as well as probable problems with generalizing results from emulators to the real world. Further, some avenues are suggested for further research. These involve assessing SDN in diverse scenarios and increasing interoperability with legacy networks to guarantee smooth migrations.

Kaczmarek et al. in [60] discussed several critical challenges that affect the performance of the SDN controllers and ultimately make them incapable of maintaining Quality of Service (QoS). One of the major issues to be identified was the controller response time at high traffic volumes with capacities as high as 200000 flows per second. Flow management plays an important role because the number and nature of the flows, such as Constant Bit Rate (CBR) and Variable Bit Rate (VBR) affect the performance. The literature describes various types of traffic, and each has its specific ways of QoS management. In addition, the authors suggest various mechanisms such as the Type of Service (TOS) field in IP packets that may have the ability to prioritize traffic. Most of the studies are performed based on emulated physical research testbeds with regard to various real-life scenarios, though they might not be able to capture all the real network complexities. To sum up, continuous research work is required to be performed to further improve controller performance and QoS mechanisms in an environment similar to real teletraffic conditions.

Lokuliyana et al. in [47] investigated QoS evaluation on SDN controllers is performed by a systematic methodology, focusing mainly on POX and RYU and logically aiming at three major steps: topology creation, controller implementation, and performance testing. Designs for network topologies were accomplished using Python. Each has specific implemented rules which would facilitate the comparison in performance between the two controllers. Test cases have been conducted using Mininet and Miniedit in a virtual environment. Communication has been made through OpenFlow, while QoS metrics such as throughput, delay, and jitter were analyzed. Possible weaknesses include the fact that the testing has been conducted in a controlled environment and, therefore, may not fully represent real-life conditions. The study narrows down to only two controllers which limits the generalization of the study. Future work may extend this study by testing more SDN-based controllers, on-field testing, and also integrating newly developed traffic engineering and advanced load-balancing techniques for further optimization of performance optimization of SDN.

Yang et al. in [14] analyzed the security issues that are involved with Software Defined Networking (SDN), as well as the challenges that are related to network connection failures. There is a fundamental problem with the location of the SDN controller, which encompasses both single-link and multi-link failures. A heuristic approach was proposed by the authors to handle the issue of controller placement for single-link failures that were accomplished via the use of GPA. Moreover, they use a Monte Carlo Simulation in combination with GPA to lessen the amount of computational work. The results of the experiments showed that the Monte Carlo Simulation and the Generalized Principal Approximation (GPA) are both reliable and efficient methods for addressing the controller placement issue in the presence of single-link and multi-link failures. Topologies like Internet2 and the Internet Topology Zoo were used to evaluate the proposal. When compared to the ideal method, the results of the simulation show that the heuristic approach is better in terms of performance while also saving a significant amount of time.

Eissa et al. in [6] investigated the software-defined networking (SDN) and the OpenFlow protocol, focusing on the SDN architecture and the OpenFlow standard. The authors study the principles behind the design of SDN architecture and the specific messages exchanged between the Ryu controller and OpenFlow switches. The paper explores the utilization of SDN, including how each switch shares the overall traffic and the use of programming languages like Pyretic, Python, and Frenetic for SDN. The simulation is conducted using a network emulator called Mininet and the Ryu controller, with Python scripts used for implementation. The authors also discuss the capabilities of switches, such as the ability to push/pop tags and adjust header field values in packets. The paper emphasizes the use of the Ryu controller software environment to build network applications that can run with Mininet. Overall, the paper provides a detailed exploration of SDN architecture, the OpenFlow protocol, and their implementation using Mininet and the Ryu controller.

Sherwin et al. in [26] provided a survey of research on Software Defined Networking (SDN) for data center network management. Because SDN allows programmatic control over network design, it is necessary for fulfilling the demands of contemporary data center network administration. The application of SDN to the management and operation of data center networks which are dynamic due to shifting resource requirements and a range of traffic profiles is the main topic of this article. VLAN functionalities, such as rewriting VLAN tags as packets traverse a network or determining VLAN membership based on arbitrary criteria can be used with SDN. SDN reduces temporary blockage in WAN links and Internet-based connections between data centers and allows for quick response in the event of a link failure. SDN gives network device monitoring flexibility by enabling dynamic switch resource adjustments and adjusting monitoring frequency and granularity by needs.

Mokoena et al. in [49] explored software-defined networking (SDN) using the OpenFlow protocol and found improvement in network management. It was deployed by using graphical simulators and virtual machines to establish network architecture in a centralized manner by utilizing Git and Ansible. This system of simulation experiment is presented with improved efficacy and adaptability compared to conventional OpenFlow techniques. The prior study encompassed research on OpenFlow-based

network management with the new approaches, SDN in IoT device management, the quality of service, and traffic management. The study further suggested an algorithm that combines SDN and OpenFlow to facilitate network management more smoothly and accurately. Simulations were conducted using GNS3, Oracle virtual machines, and Mininet VMs to deal with different scenarios. The process required assembling Cisco switches and routers through SSH and controlling them through Ansible and Git for control of versions. The good aspects that are associated with the SDN (OF) methodology were used to make it clear because of its speed and value in evaluating state-of-the-art networking management frameworks.

Kim and Feamster in [1] demonstrated software-defined networking (SDN) as a transformative approach to network management, advocating for the separation of the data plane and control plane to facilitate centralized control of network behavior. Addressing current issues in network configuration and management, the paper proposes enhancements to various aspects of network management. Notably, it explores the Procera policy language, employing functional reactive programming (FRP) for operators to articulate intricate and responsive network policies in a simple, declarative manner. Through prototype deployments in campus and home networks, the paper demonstrates the viability of Procera and underscores how SDN can enhance common network management tasks. The study highlights the importance of continuing study and improvements in SDN protocols and interfaces to address scalability issues and enable a wider range of packet forwarding operations. It offers useful insights into the problems of network management and presents SDN as a possible solution. It establishes a basis for future research efforts and practical applications in the domain of network administration and configuration.

Rasool et al. in [25] examined the classic SDN system and its associated problems along with offering a thorough overview of network management in Software Defined Networks (SDNs). In this research, several advantages of SDN are highlighted including performance, security, high availability, scalability, flexibility, and reliability. The paper talks about the fundamentals of SDN network management, the general architecture, and the primary SDN issues. Also, the study highlights the differences between SDN and conventional network management tasks. It also highlighted the importance of intelligent planning techniques for SDN virtualization. The study also mentioned the significance of managing the Operations, Administration, and Management (OAM) activities in SDN. The study offers an all-encompassing examination of network management within Software Defined Networks (SDNs), including various aspects of technology, barriers, and possible resolutions.

Bwalya and Zimba in [17] proposed an SDN approach that utilizes Ansible and Git in a centralized network architecture to address network management challenges in traditional networks. It shows how an Ansible playbook dynamically configures several devices and pushes backup files to Git for version control which can be used to do network management. The study highlighted the inefficiencies, configuration drift, and error-proneness associated with manual network management using command-line interfaces (CLI) on traditional networks. The study presents the efficacy of the suggested approach which can increase network management efficiency and decrease configuration drift in traditional networks. The suggested method boosts produc-

tivity, decreases configuration drift in network management, and lessens the need for manual CLI interactions.

Gante et al. conducted a study in [2] that investigated how wireless sensor networks (WSNs) exploit software-defined networking (SDN) as a potential solution to overcome the existing difficulties in managing WSNs. This method has significant benefits, including as improved energy savings, advanced network management capabilities, and the capability to implement new programs and features instantly. The integrated intelligence of controllers for SDN offers a complete and global perspective of the network, allowing for intelligent administration and successfully addressing limitations in traditional WSNs.The ability of SDN to simplify lower-level operations enables effective administration of network services and effortless implementation of new protocols and applications. though, there is continuing study and inquiry into fully evaluating the effect, constraints, and potential obstacles of using SDN in WSNs.

Vijay et al. in [67] examined how to manage extensive IoT networks by employing the SDN approach and a centralized hierarchical tree topology in terms of scalability and fault tolerance. LANs cannot address smart city capacity requirements, but TCP provides congestion control, reliability, scalability, and single-controller problems. Due to SDN's centralized and programmable approach, it is offered as a solution to facilitate basic network management and configuration. Hierarchical tree topology in IoT networks helps to increase a degree of fault tolerance because different devices can have multiple paths to communicate, and makes the management of all devices easier. Performance measurement with the help of tools such as Mininet or Iperf shows round-trip time or percentage of Lost packets. Also, it helps the network layers to communicate with southbound and northbound APIs to enhance its management. Lastly, the authors emphasize the need for more studies on evaluating SDN performance in various IoT scenarios especially with regard to node addition or removal they propose SDN and hierarchical topologies as potential solutions to enhance large-scale IoT networks.

Hasan et al. in [11] presented testing scenarios using the Mininet emulator along with the Floodlight controller to test the performance of an SDN simulation environment. The scenarios involve choosing bandwidth and monitoring throughput to compare the output results from each scenario. The Floodlight controller was selected for the tests due to its stability, reliability, resource usage, and good performance compared with other controllers. Overall, SDN offers benefits such as decoupling network control from data-forwarding devices simplifying network manipulation, reducing costs, and enabling greater capacity. It also opens up new possibilities for network design and strategies.

Bulbul et al. in [31] proposed an SDN-based dynamic path reconfiguration algorithm for time-sensitive networking (TSN) to address the problem of resource degradation and sub-optimal flow assignment over time. They combined Software-defined Networking (SDN) with TSN to enable seamless migration of time-sensitive flows. As they formulated the problem as an optimization problem, they evaluated non-identical dynamic path configuration strategies under deterministic communication

requirements. In worst-case conditions, the simulation findings show how changing flow assignments may increase the total number of flows included in the network and decrease the speed of time-sensitive flows. It has been observed that periodically rearranging the flow assignments can enhance the delay of time-sensitive flows and augment the quantity of flows integrated into the network. Additionally, alternative path configuration methods may include more flows into the network without introducing extra delays to the time-sensitive traffic.

Islam et al. in [9] evaluated the performance of SDN controllers in a wireless network. The authors examine popular SDN controllers like ONOS, POX, Ryu, and Floodlight in an emulated wireless network. They use tools like Mininet for simulation and iperf for generating traffic. They provide observations on the efficiency of the examined SDN controllers in the wireless network. The paper addresses the need for studies on the performance evaluation of SDN controllers in wireless networks. The observations from this study can be valuable for selecting the appropriate SDN controller for wireless networks. Also, this paper mentions previous studies that have evaluated SDN controllers in wired networks, providing comparisons between controllers like Floodlight, POX, Ryu, and Pyretic.

Ahmad and Mir in [15] discussed the different deployment models for SDN control planes, including physically centralized, physically distributed but logically centralized, and hybrid architectures. The paper also discusses the advantages and disadvantages of distributed SDN control planes, as well as research challenges including figuring out how many controllers are needed, accomplishing dynamic load balancing and concurrent consistent state sharing, and placing distributed controllers in various network scenarios. Different distributed SDN controllers are examined. The study also examines different centralized SDN controllers in terms of performance metrics and talks about centralized SDN control planes. Based on the number of concerns addressed, it categorizes the processes employed by SDN controllers to solve scalability, dependability, consistency, and security, and labels them as good, limited, or very limited. The study also discusses unresolved problems and research obstacles in various SDN control planes.

Alssaheli et al. in [30] addressed the escalating demand for internet services, emphasizing the need for agile load-balancing mechanisms to replace traditional, inflexible, and costly solutions. They advocate for Software-Defined Networking (SDN) as a transformative alternative, separating network control, applications, services, and forwarding roles for enhanced flexibility, cost-effectiveness, and programmability. The study specifically introduces SDN-based Load Balancing, utilizing predefined servers in the server farm to process Internet Protocol (IP) data packets efficiently. Assessment metrics, covering throughput, delay, and jitter across scenarios (A, B, and C), reveal SDN's positive impact on load balancing and heightened network performance. It highlights the effectiveness of SDN controllers, highlighting enhanced scheduling features and robustness in the face of fluctuating client counts. The potential of SDN load balancing to effectively control network traffic and provide strength, security, and scalability is also illustrated. Proactive monitoring tools provide administrators the ability to evaluate server and balancer status, selectively stop functionality for certain problematic servers, and improve user experience. Fu-

ture studies should investigate AI-based techniques to simplify the deployment of SDN load-balancing, enabling routing with AI merging and autonomous management.

Saxena et al. in [53] provided an overview of software-defined networking (SDN) and its impact on network load balancing. It discusses the challenges of uneven load distribution in SDN and the need for load-balancing techniques to improve network performance. The paper reviews existing SDN load-balancing techniques and their contributions, critically analyzing their features, advantages, and limitations. It identifies different classification criteria for SDN load-balancing methods and their impact on overall load-balancing performance. The paper also highlights the performance metrics for evaluating these load-balancing algorithms. It suggests possible future research directions to enhance load-balancing techniques in SDN environments and provides a road map for researchers in this field.

Tanguturi.R.C. et al. in [65] investigated the vast insight into SDN compared to traditional TCP/IP architectures; hence, it shows the main advantage of SDN, which is lower latency with better bandwidth management because of central control and programmability. Performance improvement in SDN is demonstrated within a Mininet emulator, but there is no mention of scalability or applicability in real environments. Other common concerns include performance mismatch when scaled to larger, more complex networks and excluding real-world environmental factors, such as hardware differences and external traffic conditions. Regarding future work, the paper will include a more comprehensive performance metric than just latency, jitter, and bandwidth, with real-world testing to validate the simulation results. This will, in addition, be useful in the study of hybrid architectures that integrate SDN and traditional TCP/IP to exploit the respective strengths in arriving at more flexible networking solutions. These considerations put into perspective some of the important areas of continued research in network performance analysis.

Caoand and Rong in [57] discussed the performance of TCP in SDN through bandwidth and latency, using Mininet for simulation. This review determines that the bandwidth from TCP decreases with the increased number of switches. Multithreaded transmission modes are time-effective compared to single-threaded, though they introduce complexities related to thread synchronization and possible network congestion. Most of the research focused on bandwidth, while other important metrics such as latency and packet loss were not studied. This dependence on simulations also binds these results to/Applications in actual SDN deployments. Future research is expected to be performed within a wider range of various performance metrics, including controllers and hardware, which influence end-to-end network optimization.

Noman et al. in [12] analyzed the performance of the POX controller carried out in the context of the SDN framework over the OpenFlow 1.0 protocol to enable the data control plane interface. Network emulation was performed with Mininet; network performance characteristics such as bandwidth consumption, service delay, and loss have been obtained with Iperf and D-ITG tools. The performances of the results revealed that the controller effectively controlled the bandwidth up to 500Mbps and

the CPU load of the controller was between 70-80%. Packet loss decreased with increasing distance up to eight hops However, fairly poor results were observed at higher packet sizes, indicating problems with scalability and reliability. It is crucial to enhance the capacity of POX so it supports the big data packet, besides to recommend other protocols and controllers for stronger SDN implementation.

Lunagariya et al. in [22] presented the adaptive load balancing techniques for multi-SDN controllers that present significant advancement in enhancing network performance. As it is with data links and server load balancers, the algorithms deployed in SDN networks reduce the latency levels hence improving responses. Hybrid routing algorithms enhance dynamic server load distribution to optimize whereas centralized controllers get rid of data plane congestion. Multi-path strategies that are pre-emptive enhance flow distribution and different methods of load balancing such as IP Hash and Least Connection demonstrate different results in performance with IP Hash performing much better than traditional methods. Also, heuristic strategies for control of switch migrations have been developed and dynamic loading schemes improve link usage considering the priority of the flow. In summary, the review of existing works points out the focus on the process and variability in the load distribution and a solid foundation for future studies in the field of SDN.

Rahman et al. in [52] presented the performance evaluation of three SDN controllers, namely, NOX, POX, and RYU, using linear and tree topologies based on metrics like Flow Setup Latency, Throughput, and TCP/UDP Bandwidth. Simulation results show that RYU is consistently outperforming NOX and POX while maintaining stability under heavy traffic whereas in the case of NOX and POX, performance degradation under load is at a significantly higher level. Topology affects performance too. Above four hops, TCP bandwidth halves, while latency rises sharply. However, limiting this study and evaluation to just linear and tree topologies, and only three controllers, does not allow for generalization of the findings. This work will overcome these limitations by extending the studies to complex topologies like Fat Tree and Spine-Leaf, also exploring other parameters such as topology discovery time and multi-threading.

Ram et al. in [62] analyzed the performance of two widely used SDN controllers, POX and Ryu, based on the Mininet-Wi-Fi emulator. They evaluate different metrics such as jitter, throughput, packet loss, and delay using the Distributed Internet Traffic Generator. This work represents unique network performance with respect to wired and wireless environments, using Fast Ethernet for the wired connections and evaluating packet sizes from 128 to 1,024 bytes. The authors compare several network topologies: single, linear, and tree structures. Indeed, the results clearly show that Ryu has always outperformed POX in latency, packet loss, and jitter, whereas it is distinctively superior in terms of throughput, particularly in wireless scenarios.

Ali and Amin in [39] reviewed the ways of improving the integration of Software-Defined Networking (SDN) into large-scale Wi-Fi networks. The proposed study evaluated the results and shows how SDN uses bandwidth, round-trip time, and jitter to support concurrent access points and users while also highlighting the prob-

lems faced by traditional Wi-Fi systems, especially in developing the IoT. With the enhanced experimental setup of Ryu Controller along with Mininet-Wi-Fi, the integral features of SDN are evaluated demonstrating that the implementation of Wi-Fi network management has significant potential with SDN solutions as compared to the conventional systems for Wi-Fi. Lastly, the paper recommends SDN as a solution to Wi-Fi scalability problems and provides further areas of research in wireless networking.

Iqbal et al. in [8] addressed security concerns with Software Defined Networking (SDN) and offered possible fixes to improve the SDN controller's security architecture. They tackle security concerns related to SDN, concentrating on weaknesses such as DDoS, DoS, and application violence. For GUI protection, the authors stress the significance of SSL/TLS integration, authentication, and logging/security. The authors investigate improving data security in the SDN environment by utilizing encryption methods (DES and AES) and role-based authorization (FortNOX). The paper highlights the serious consequences of hacks while examining internal and external threats in SDN. Attack graphs and alert correlation models evaluate security, emphasizing hazards associated with false traffic flows and vulnerabilities in SDN downstream APIs such as OpenFlow. These attacks can be initiated by malicious or non-malicious devices to deplete resources in controllers or forwarding devices. The paper explains SDN security issues and suggests ways to improve SDN controller security.

Yurekten et al. in [28] conducted a thorough literature review to create a taxonomy of SDN-based defenses against typical attack types and group suggestions according to cyber threat classification. In software-defined networks, the study classifies cyber threats and analyzes network defense strategies used to counter them. The study reviews and discusses approaches, tactics, and processes for SDN-based attack prevention and mitigation proposed in the literature, categorizing them based on cyber threat types. The evaluation lists protection strategies for several threat categories, including web application attacks, malware and social engineering attacks, and sniffer attacks. The study also looks at how SDN-based defense techniques might be utilized with emerging technologies like 5G and IoT to improve security, and it emphasizes the need for more research to solve gaps in the existing literature.

Yungaicel et al. in [27] targeted networks such as 5G communication networks, Internet of Things (IoT), and DDoS attacks have become more complicated and extensive. Previous research has proposed methods for identifying DDoS attacks. However, most of the studies did not take advantage of the most recent datasets. This design makes use of several machine learning (ML) and deep learning (DL) models along SDN architecture to recognize DDoS attacks at both the transport and application layers. The efficiency of the authors' detection system in recognizing transport and application-layer DDoS attacks with a high degree of precision was simulated using Mininet and the ONOS SDN controller. The K-Nearest Neighbors (KNN) approach, which is a machine learning technique, has shown effectiveness in slow-paced attacks. LSTM, GRU, and KNN have shown greater performance when it comes to detecting attacks at a slower rate. To be more specific, LSTM was able to achieve an average detection rate of 85% for all slow-rate attacks when the attack

rate was 300 connections per second.

Maleh et al. in [48] provided an overview of SDN security, putting the study literature into a taxonomy that emphasizes the key characteristics and contributions to various SDN architecture layers. The authors draw attention to how security concerns in SDN differ from those in traditional networks in that they have a greater effect on centralized controllers due to denial-of-service assaults. They also talk about how difficult it is to secure the SDN architecture itself. The study highlights the importance of additional research in this field and points out important knowledge gaps. Discussions of SDN vulnerabilities, attack types, and mitigation strategies are covered in the research study. Examples of these include flow table size limitations and the absence of encryption in communications between SDN switches. To confirm the legitimacy of flow rules and identify unlawful flow rules made by malicious apps, the authors also discuss the use of authentication and authorization models, such as PERM-GUARD.

Shaghaghi et al. in [13] focused on securing the software-defined network platform itself, rather than enhancing existing network security services or creating new security services. The study highlights the difficulties and weaknesses in SDN data plane security. It talks about the dangers of compromised forwarding devices and emphasizes how important it is to secure the software-defined network infrastructure itself. The importance of the northbound API for creating apps and network services inside the SDN architecture is also mentioned in the article.

Chasaki et al. in [19] discussed the issue of malicious hosts executing malicious programs that avoid SDN-based detection features. The suggested SDN security solution utilizes machine learning classifiers to learn from previous system behaviors, examines the system call utilization of various installed SDN applications regularly, and reliably discovers strange activity or malicious apps. The only methods available to access the kernel are through system calls, which offer an in-depth knowledge of how a network node processes data. The research employs a form of "side-channel analysis" to find any malicious software or activity that influences the virtual host's processing activity. The virtual host's OS calls are the selected side channel. Machine learning classifiers are trained with recovered systems to call out various malicious and benign app features to precisely identify suspicious behaviors that alter the host's processing behavior.

Abdulqadder et al. in [3] presented a robust security scheme for 5G networks, incorporating Software Defined Networks (SDN), cloud computing, and Network Function Virtualization (NFV). The scheme includes a very creative Secured Authentication and Handover Mechanism (HS-AOHM) for mobile users, minimizing the handover latency without compromising user privacy. They introduced a Switch Assignment (TBSA) algorithm to mitigate the overloading attacks of the flow table by attributing packets to underload switches. Also, the DDoS attacks are discovered using entropy analysis at the controller, and any fishy packets are routed to a cloud-based scrubbing Virtual Network Function (sVNF). To distinguish between suspicious and malicious packets, the sVNF drops the latter and classifies suspicious packets using a Hybrid Fuzzy with Artificial Neural Network (HF-ANN) classifier. Security pa-

rameters such as changeover detection accuracy, latency, switch failure rate, holding time, and delay are improved, as shown by extensive simulations. The study highlights the benefits of the suggested design by contrasting it with other cutting-edge developments. The suggested strategy outperforms earlier techniques with its high detection accuracy and short latency. The relevance of security in 5G networks and the necessity of adapting to changing user needs are emphasized in the study.

Krishnan et al. in [46] aimed to use OpenStackDP, a security framework that can identify abnormalities, lightweight monitoring, a threat analytics engine that makes use of machine learning and artificial intelligence techniques, and defensive measures that are implemented as virtual network functions (VNFs) are the components that make up the framework. The framework makes use of powerful streaming analytics methods, technologies, and machine learning capabilities. The framework's multiphase collaborative anomaly detection technique has an accuracy of 99.81%, with average latencies of 0.27 milliseconds and reaction times of less than nine seconds. Additionally, the framework's response times are short. Based on the findings of the simulations and analysis, it has been determined that the OpenStackDP network analytics framework is superior to the SDN-based OpenStack solutions that were previously used for cloud designs in terms of both security and performance.

Bakhshi in [4] established security criteria, attributes, and mitigation strategies while highlighting security considerations in Wireless Software Defined Networking (WSDN) and classifying and addressing major threats. To guarantee the authenticity, privacy, consistency, availability, and integrity of control and data plane entities in wireless networking, it emphasizes the vital necessity for strong countermeasures. In light of their ability to impede network traffic and jeopardize controller and control channel traffic, possibilities for attacks on forwarding and end-user devices are also discussed. Despite these difficulties, the study focuses on the benefits of a centralized WSDN architecture, highlighting real-time programmability and worldwide traffic monitoring capabilities. By addressing security flaws and guaranteeing attack resilience in WSDN deployments, the paper acknowledges the difficulties associated with wireless media, including latency, jitter, and communication delays. It also provides a thorough framework for further research and development in this rapidly developing field.

Khairi et al. in [20] discussed the application of machine learning (ML) algorithms for the detection and classification of flow conflicts in Software-Defined Networking (SDN) emphasizing the significance of intelligent and interpretative conflict classes that can hurt the performance of the network. Several ML algorithms, including Decision Tree (DT), Support Vector Machine (SVM), Extremely Fast Decision Tree (EFDT), and a hybrid DT-SVM model, were evaluated for efficiency in enhancing detection accuracy and efficiency in SDN networks. The experiments were conducted on Fat Tree and Simple Tree topologies in Mininet simulations with a dataset of 1,000 to 100,000 flows. The results showed that EFDT achieved the highest detection accuracy of 99.49%, while the hybrid DT-SVM model also achieved satisfactory performance with 99.27% accuracy. In addition, 95.73% accuracy had been achieved by the EFDT algorithm in the classification of conflict flow types. These enhancements had shown significant improvements compared to traditional

models, including DT and SVM, in flow conflict detection.

Salau et al. in [63] focused on SDN-based network traffic classification using machine learning techniques. It highlighted the increasing risk to user privacy due to encrypted traffic, which has made precise identification necessary. It was discussed by ByteSGAN, a GAN-based semi-supervised learning method integrated into the SDN Edge Gateway to optimize network resources, although it did not address data imbalance issues. Various machine learning models like Decision Tree, K-NN, etc. had been utilized for the classification of traffic types, among which the Decision Tree model achieved the highest accuracy of 99.81%. Traffic classification has been improved using SDN and machine learning, particularly in the detection of encrypted packets and QoS enhancement. Evaluation metrics like accuracy, precision, recall, and F1-score were used for the evaluation of the models. Traffic simulation and network design were achieved with the use of D-ITG and Mininet tools. Limitations of the review were that it was challenging with encrypted traffic and failed to balance data in the GAN approach.

Nunez et al. in [50] showed 15 extracted features such as packet count, packet transmission time, etc., the paper indicated that 15 features including packet count and packet transmission time, were used to identify traffic flows for multiple protocols like DNS, WWW, FTP, ICMP, P2P, and VOIP. The methodology includes data collection from an SDN controller in real-time, data preprocessing, and training machine learning models like DT, RF, KNN, SVM, LR as well as NB. Other models however did not perform as well as the DT classifier which resulted in an accuracy of 99.8%. Optimality of classification performance at the computational efficiency level was achieved by means of feature selection. The study establishes that accurate classification is supported by centralized traffic monitoring provided by SDN. Nevertheless, scalability, network complexity handling, and feature selection optimization come with limitations. The issue of ensemble learning and real-world deployment should be addressed for future work to improve robustness and adaptability.

Gomez.C. et al. in [42] ML models were used in an SDN-based traffic classification system with real-time traffic categorized in six protocol classes (WWW, DNS, FTP, ICMP, P2P, VOIP) and 15 extracted features, which were executed with DT, RF, KNN, SVM, LR, and NB. In precision and efficiency, DT outperformed the others who could attain up to 99.8% accuracy. However, feature selection led to improved classification, but it remained difficult to scale and work with encrypted traffic. For a second study, Mininet and D-ITG were used for simulation and supervised (DT, RF, SVM, LR, AdaBoost) and unsupervised (K-means) learning were integrated together. 99.81% accuracy was again the best for DT. PCA reduced the redundancy of features but was not able to classify similar traffic types such as Ping and DNS.

Gonzalez et al. in [43] analyzed SDN controllers (RYU, ODL, and ONOS) and machine learning algorithms to detect anomalies such as Decision Tree (DT), Support Vector Machine (SVM) and Neural Networks (NN) for real-time traffic classification. DT has 100% accuracy on ODL and RYU, and ONOS scored 97%. For SVM-related tasks, RYU and ONOS achieved 94% and 92% respectively ODL excelled at 100%. At 80%, RYU was the best that NN analysis offered, but ODL outperformed all oth-

ers. Using ONOS, 95% of application layers and 1 percent of transport layer DDoS attacks were detected. The framework was simulated using the Mininet emulator, but there is a lack of testing and the focus is on throughput and latency, not scalability or security so generalizability and applicability to the real world are still concerns.

Khairi et al. in [20] discussed the application of machine learning (ML) algorithms for the detection and classification of flow conflicts in Software-Defined Networking (SDN) emphasizing the significance of intelligent and interpretative conflict classes that can hurt the performance of the network. Several ML algorithms, including Decision Tree (DT), Support Vector Machine (SVM), Extremely Fast Decision Tree (EFDT), and a hybrid DT-SVM model, were evaluated for efficiency in enhancing detection accuracy and efficiency in SDN networks. The experiments were conducted on Fat Tree and Simple Tree topologies in Mininet simulations with a dataset of 1,000 to 100,000 flows. The results showed that EFDT achieved the highest detection accuracy of 99.49%, while the hybrid DT-SVM model also achieved satisfactory performance with 99.27% accuracy. In addition, 95.73% accuracy had been achieved by the EFDT algorithm in the classification of conflict flow types. These enhancements had shown significant improvements compared to traditional models, including DT and SVM, in flow conflict detection.

Ccubukccu et al. in [58] examined how Software Defined Networking (SDN) and Network Function Virtualisation (NFV) enhance Quality of Service (QoS) within 5G networks through their combined application. An SDN application had been developed within an open-source simulation environment to test two methods: The system implements traffic rerouting with Dijkstra's algorithm together with dynamic network scaling operations built from SDN and NFV technologies. Under network congestion, the research findings showed SDN and NFV together preserved consistent QoS while the dynamic scaling method demonstrated superior performance than traditional rerouting. Current performance data rates had improved notably through Flow-1 delivering 46.9 Mbps on average.

Karn et al. in [44] summarized key advancements in traffic classification and load balancing in Software-Defined Networking (SDN) quoting improvements such as dynamic load-balancing strategy on wildcard rules that achieved response time improvement in the updating of flow tables and a server notification mechanism for efficient traffic redirecting. It discussed ECMP routing that divides traffic across multiple paths but with challenges in managing large flows. The authors suggested elephant flow detection to prioritize significant traffic and reduce latency. The authors even explored machine learning models like SVM and random forests to classify traffic, though real-time classification remained an issue. The Classification and Regression Tree (CART) algorithm was found to be superior to others for real-time classification. Traffic-type-based load balancing showed performance improvement in networks but issues such as real-time classification and throughput-based balancing algorithms were mentioned, pointing out areas for further research.

Yaseen et al. in [38] concentrated on load balancing in cloud computing, emphasizing the significance of effective resource management because of the growing user base. It analyzed multiple algorithms, such as Round Robin (RR), Weighted Round

Robin (WRR), and Dynamic Weighted Round Robin (DWRR), and presented the Priority Weighted Round Robin (PWRR) algorithm as an improvement of WRR. PWRR integrated task priorities into the scheduling system, enabling more effective management of tasks according to their urgency. The findings indicated that PWRR excelled over RR and WRR in terms of execution cost, waiting time, turnaround time, and average execution duration. It also showed notable enhancements in finishing high-priority tasks when compared to DWRR. The performance evaluation was carried out using the CloudSim simulation tool, revealing that PWRR diminished the duration for high-priority tasks in comparison to other algorithms. Nevertheless, the research did not tackle the scalability of PWRR in more extensive cloud settings, and real-time dynamic load balancing continued to pose a challenge.

Chapter 3

Methodology

To perform the performance evaluation of Software-Defined Networking we used Mininet which is a tool capable of emulating virtual networks [56]. This will efficiently create network topologies of practically any size and complexity without necessarily acquiring or setting up physical hardware. RYU controller was chosen as an SDN controller for this study which was configured to handle the custom network topology that combines tree, star, linear and cluster structures for replication of real-world scenarios with various patterns of traffic including media traffic and routing complexities with flow control. Mininet and RYU were running on the same virtual machine with the purpose of simplifying the setup and making it more resource-efficient.

3.1 Experimental Setup

All the experiments were conducted on a single virtual machine hosted by a physical machine with a 13th-generation Intel Core i7-14700K processor, 16GB DDR5 RAM, and Windows 11 Pro as the base operating system. Oracle VM VirtualBox 7.1.6 running Ubuntu 22.04.4 LTS as the guest OS which is configured to efficiently run all the network simulations and other tasks required for various experiments with 10 vCPUs, 30 GB RAM, and 55GB of storage. RYU controller 4.3 was installed and configured to run on the default OpenFlow port 6633. Mininet 2.3.1b4 was used to emulate a user-defined custom topology that combined tree, star, linear and cluster topologies. The OpenFlow protocol version was set to 1.3 on all switches. Matplotlib 3.10 was used to visualize experimental results which is a tool with advanced options for plotting performance metrics such as throughput, latency, and jitter and also other media traffic metrics. The specific software versions are:

Physical Machine: Windows 11 Pro (Core i7, 64 GB DDR5 RAM)
Virtual Machine: Oracle VM Virtual Box 7.1.6
Virtual Machine OS: Ubuntu 22.04.4 LTS (10 vCPUs, 30 GB RAM, 55GB Storage)
Network Simulator: Mininet 2.3.1b4
Controller: RYU 4.3
Switch: OpenvSwitch
Protocol: OpenFlow-on Port 6633
Visualization Tool: Matplotlib 3.10

3.2 Machine Learning for Flow Classification

To enable efficient traffic management in our SDN-based network, we implemented machine learning models to classify network flows into mice and elephant categories. The dataset we used in our study was obtained from GitHub [68]. We then applied a data cleaning and filtering process to remove redundant and misleading samples and ensure a balanced representation of both types of flows. The dataset consisted of flow-based network traffic data from an SDN environment. It was then divided into training and testing sets following a standard splitting ratio of 80:20 to prevent overfitting and ensure generalisability. The train set contained 2,36,064 flows and the test set contained 59,016 flows. The label distribution within the dataset was analyzed to maintain an optimal balance between the two flow categories.

Several machine learning models were trained and evaluated including Logistic Regression (LR), Neural Networks (NN), K-Nearest Neighbor (k-NN), Support Vector Machine (SVM), and Naive Bayes (NB). These models were chosen for their ability to handle non-linear relationships in network traffic patterns and their effectiveness in classification tasks. The evaluation of model performance was performed using standard metrics such as accuracy, precision, recall, and F1-score. Among the models tested, we used **k-NN** for its higher accuracy which is shown in Figure 3.1 and also this model performed well in predicting the flow in real-time.

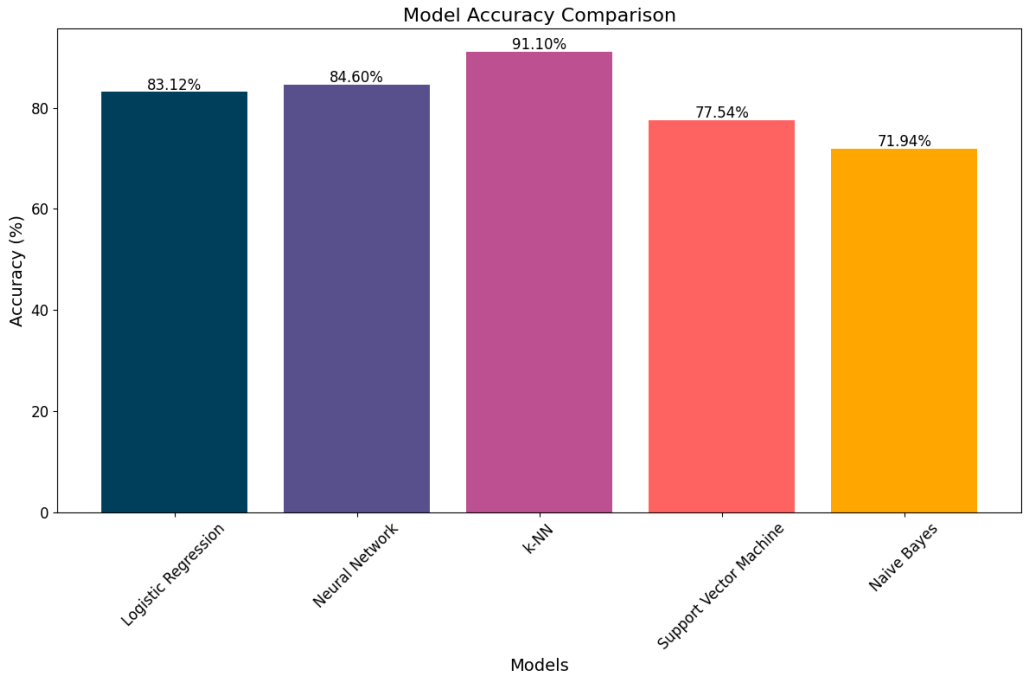


Figure 3.1: Accuracy of Used ML Models

We have chosen this model by analyzing the confusion matrix and classification report shown in Figure 3.2. It showed good precision & recall means balanced performance for both classes and the confusion matrix shows less misclassification which means fewer false positives & false negatives.

So, K-Nearest Neighbor (KNN) was integrated into the controller to predict the type of flow in real time based on extracted packet features. This classification played

```
Confusion Matrix (KNN):
 [[15105  2496]
 [ 2754 38661]]
```

(a) k-NN Confusion Matrix

```
Classification Report (KNN):
              precision    recall  f1-score   support

           0       0.85      0.86      0.85     17601
           1       0.94      0.93      0.94     41415

    accuracy                           0.91     59016
   macro avg       0.89      0.90      0.89     59016
weighted avg       0.91      0.91      0.91     59016
```

(b) k-NN Classification Report

Figure 3.2: Confusion Matrix and Classification Report

a crucial role in the load-balancing mechanism as elephant flows required a more structured handling approach to prevent congestion whereas mice flows followed a traditional forwarding mechanism.

3.3 RYU Controller Modification

The Ryu controller was chosen for its lightweight, event-driven architecture and native support for OpenFlow which enables fine-grained control over flow entries and traffic forwarding policies. Ryu's modular nature allowed seamless integration of our time domain Round Robin scheduling algorithm for load balancing and the machine learning-based flow classification model. Furthermore, Ryu's multi-threading capabilities enabled the Round Robin scheduler to operate as an independent thread, ensuring efficient queue management without disrupting the controller's core event loop.

The point to note here is that our proposed Round Robin algorithm is of a different approach than other Round Robin algorithms used for load balancing in SDN environments. The proposed algorithm is derived from the algorithm used in CPU scheduling. That is, it also works on a set time quantum. After each time quantum the controller moves down the queue and sends segments from the next packet. Also to ensure the reliability of the Round Robin algorithm, we defined a separate thread for the Round Robin algorithm itself which will ensure that round robin is not dependent on the controller and is able to function and process queued packets even if the controller is not actively participating in the load balancing process.

To implement this approach we modified the default *packet_in_handler()* to align with our requirements. Upon receiving an incoming packet, the controller extracts essential Ethernet and packet-level features determining whether the flow belongs to ARP, ICMP, TCP, or UDP traffic types. Once determined, the controller applies the machine learning model to determine whether the flow qualifies as an elephant or mice flow. If a flow is classified as an elephant, at first it is segmented into smaller chunks and added to the queue along with their desired datapath, in-port, and the qid. After that, it is processed by the Round Robin scheduler to ensure efficient bandwidth utilization. Similarly, if the flow is categorized as mice then it is either forwarded directly based on existing flow entries or subjected to a flooding mechanism depending on the traffic type and network conditions. The ML approach in the controller is described in Algorithm 1 below where the **k-NN** model is implemented.

Algorithm 1 ML Implementation in RYU Controller

```
Initialize RoundRobinController
TIME_SLICE ← 0.2                    ▷ Time slice for round-robin scheduling (seconds)
MAX_QUEUE_SIZE ← 1000               ▷ Maximum packet queue size
OFP_VERSIONS ← [ofproto_v1_3.OFP_VERSION]
SEGMENT_SIZE ← 512
procedure INITIALIZE
    Super Constructor: SUPER(RoundRobinController)
    available_ports ← {}            ▷ Key: datapath_id, Value: list of active ports
    packet_queue ← []               ▷ Queue of packets for round-robin scheduling
    round_robin_index ← -1          ▷ Current index for round-robin
    qid ← 1
    scheduler_thread ← HUB.SPAWN(round_robin_scheduler)
    mac_to_port ← {}                ▷ MAC-to-port mapping for hosts
    packet_queue ← []               ▷ Queue of packets for round-robin
    base_path ← Os.Path.Dirname(Os.Path.Abspath(__file__))
    model_path ← Os.Path.Join(base_path, 'KNN.pkl')
    LOGGER.INFO('————————————————————')
    LOGGER.INFO(model_path)
end procedure
```

The entire working flow of our modified controller is represented in a structured diagram in Figure 3.3, outlining the decision-making process from packet arrival, feature extraction, flow classification, and subsequent routing strategy selection. This approach effectively balances traffic loads, reduces congestion, and optimizes network performance within the SDN framework.

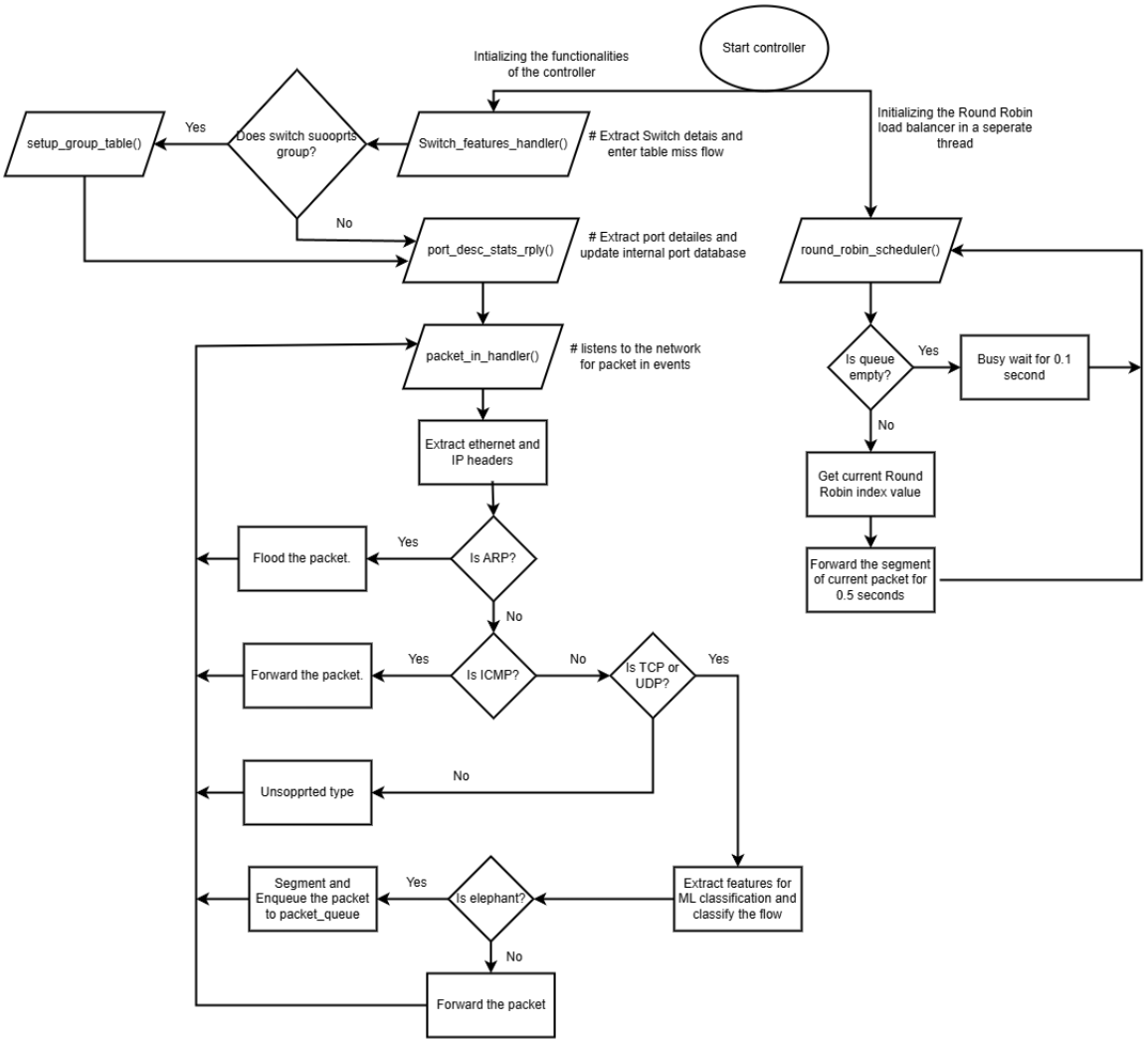


Figure 3.3: Working Flow of Our RYU Controller

3.4 Network Topology Design

The designed network topology is a combination of four subtopologies: Tree, Star, Linear and Cluster topologies. All the sub-topologies are connected with a Root Switch. Each sub-topology was chosen in order to simulate various real-world network configurations, while all of them combined to provide a varied testbed for performance evaluation under SDN control.

1. **Tree Topology**: The Tree Topology was designed to capture the hierarchical structure commonly found within data centers. It has one root switch with multiple layers of switches branching out in a hierarchical manner. The primary tree switch (*ts1*) was connected to two sub-switches (*ts2*, *ts3*) and ts2 had an additional child switch (*ts4*). Each switch had multiple hosts connected to it except the root one. The *ts4* had also two additional hosts (*th9*, *th10*) connected to it. This structure mimics the core-aggregation-access layers in large networks [52]. The tree topology in our setup consisted of 4 switches and 10 hosts.

2. **Star Topology**: The Star Topology emulated a centralized network design which is commonly seen in local area networks (*LANs*) [52]. A single central switch (*ss1*) was used to connect multiple hosts. The central switch was directly connected to the root switch (*rs1*) which ensured a single point of control with the controller. The hosts sh1 to sh5 were connected to the *ss1* switch which forms a star structure. This topology is particularly useful for evaluating centralized traffic flow and bottlenecks.

3. **Linear Topology**: The Linear Topology represented a chained network structure which is commonly found in backbone networks or point-to-point communication models [52]. It consisted of four switches (*ls1, ls2, ls3, ls4*) connected in series in which the first switch (*ls1*) was linked to the root switch (*rs1*). Each switch in this chain had a dedicated host (*lh1* to *lh4*) which made a total of 4 hosts. This topology helps in evaluating delay and performance across multi-hop networks.

4. **Cluster Network**: The Cluster Network was introduced in this upgraded topology to simulate an isolated sub-network that operates under central control. A dedicated switch (*cs1*) was added and connected to the root switch (*rs1*). This switch had three hosts (*ch1*, *ch2*, *ch3*) which is directly connected to *rs1*. This setup represents a separate organizational network such as a research lab or a specific department within a larger network.

5. **Root Switch and Centralized Control**: Unlike the previous topology where the controller was connected to multiple switches this newly upgraded topology introduced a **root switch (*rs1*)** which acts as the only connection point to the SDN controller (RYU). All sub-topologies (Tree, Star, Linear, and Cluster) are connected to this root switch. This structure reduces controller overhead and makes decisions centralized also making it easier to manage network traffic efficiently.

The upgraded network topology now consists of 22 hosts and 11 switches which significantly increases the scale and complexity compared to our previous implementation. The introduction of a cluster-based topology and the root-switch-based hierarchical structure makes the network more adaptable to real-world scenarios. These sub-topologies include cloud computing, enterprise networking, and SDN-based traffic engineering. The entire network is managed through an RYU controller which is connected exclusively to the root switch. Communication between the controller and the switches is being established using the OpenFlow 1.3 protocol which ensures centralized and efficient traffic control. This design allows for efficient load balancing, better fault tolerance, and improved network flow optimization.

3.5 Topology Creation Process

3.5.1 Writing the Python Script

To build the topology we wrote a Python script called ***newtopo.py*** which describes every element of the network. This script was manually developed using the Mininet Python API in which explicit declaration of switches, hosts, and links were performed. In this way, a very detailed and customized setup of the topology was represented. The Python script allowed for flexibility in the following:

- Configuration of host IP addresses and MAC addresses.
- Configuration of links between switches and hosts.
- Introduction of the RYU controller as a remote OpenFlow controller for the switches.

Algorithm 2 presents a partial pseudocode representation of the centralized custom topology construction.
The Python script was executed in the Mininet simulator to build the network and allow dynamic traffic generation for performance tests and analysis. In the script, to define each network component, *addSwitch()* and *addHost()* functions were used. Also to create links between switches and host, *addLink()* was used. Finally, the script connects all sub-topologies into a unified topology.

3.5.2 Visualizing the Topology in MiniEdit

Once the Python script was done, we used MiniEdit in order to be able to visually see the network topology. MiniEdit is a GUI tool for Mininet that gives us a better chance to look at the layout of the network and assure ourselves that the topology is indeed what we want to design [56]. In order to visualize the final Python-based topology, we designed it including Tree, Star, Linear and Cluster topologies, connected with links as shown in Figure 3.4.

Algorithm 2 Pertial Python Script for Centralized Custom Topology

```
procedure CentralizedCustomTopo
    Initialize Topology
    root_switch ← AddSwitch('rs1')
    tree_switch1 ← AddSwitch('ts1')
    tree_switch2 ← AddSwitch('ts2')
    tree_switch3 ← AddSwitch('ts3')
    tree_switch4 ← AddSwitch('ts4')
    AddLink(root_switch, tree_switch1)
    AddLink(tree_switch1, tree_switch2)
    AddLink(tree_switch1, tree_switch3)
    AddLink(tree_switch2, tree_switch4)
    for i ← 1 to 8 do
        host ← AddHost('th' + i)
        if i ≤ 4 then
            AddLink(tree_switch2, host)
        else
            AddLink(tree_switch3, host)
        end if
    end for
    th9 ← AddHost('th9')
    th10 ← AddHost('th10')
    AddLink(tree_switch4, th9)
    AddLink(tree_switch4, th10)
end procedure
```

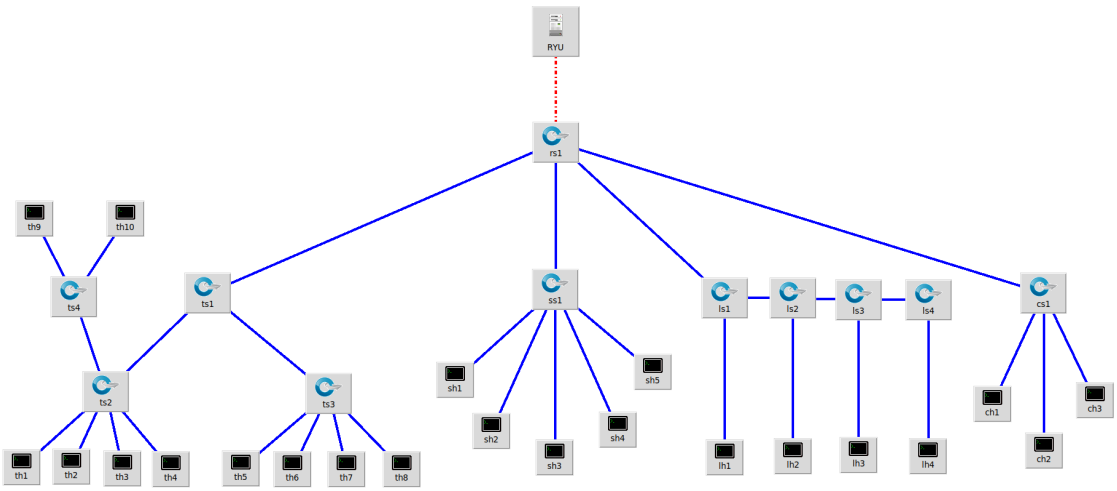


Figure 3.4: SDN Custom Topology

3.5.3 Running the Topology in Mininet

After viewing and confirming that the topology was correct in MiniEdit, we brought the topology up in Mininet using the following command:

sudo python3 newtopo.py

This script created the topology and established the links between the hosts and switches. It also connects all switches to the RYU controller running on the same virtual machine as given by Figure 3.5.

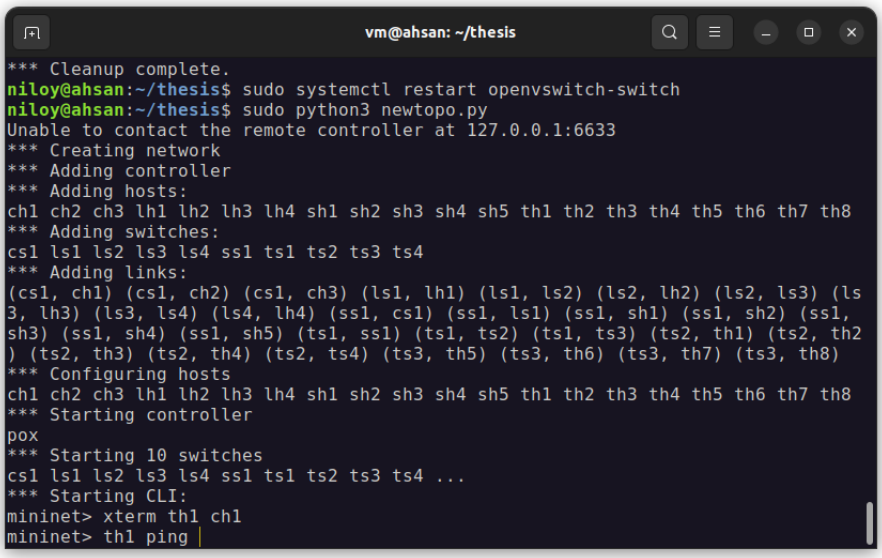


Figure 3.5: Running Topology in Mininet CLI

The connectivity between the hosts across Tree, Star, Linear, and Cluster topologies was ensured using Mininet CLI. In Figure3.6, all the nodes are able to ping each other meaning they are active and reachable.

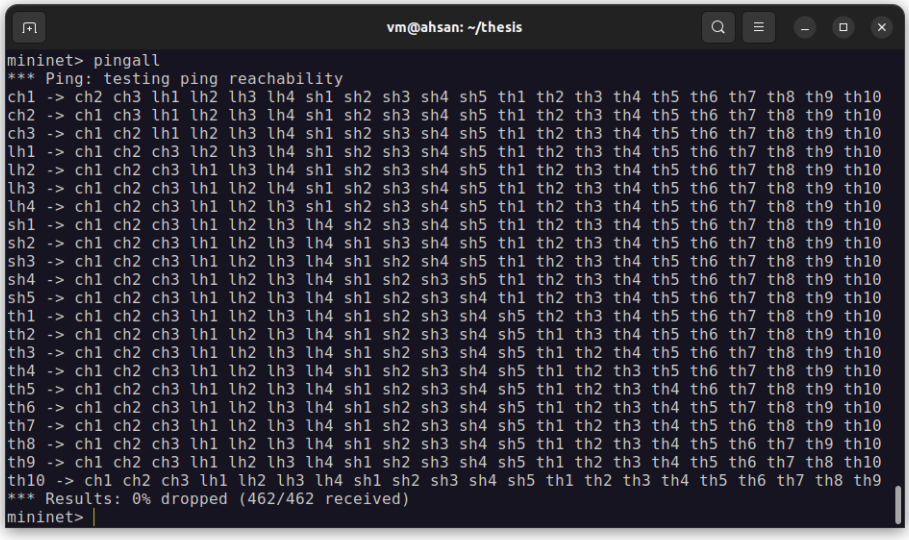


Figure 3.6: All nodes successfully reachable

3.5.4 Integration with RYU Controller

The proposed RYU controller is taking part in network management, optimization, and load balancing on the default OpenFlow port of 6633. The controller is responsible for installing the flow rules in a dynamic pattern, reacting to the network traffic created between hosts acting as a client-server pair [66]. The following command was executed to turn on the RYU controller also shown in Figure 3.7

ryu-manager/RYU/ryu/ryu/app/Roundrobin_LB_withML.py

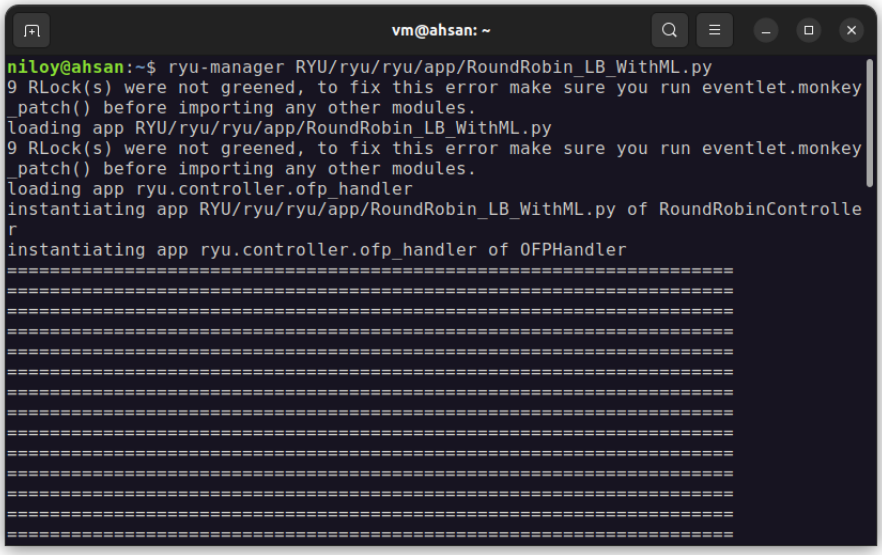


Figure 3.7: Starting the RYU Controller

This enabled the controller to learn about the network topology and install flow rules automatically, which we later analyzed using the Mininet **dpctl dump-flows -O openFlow13** command in order to view the flow entries shown in Figure 3.8

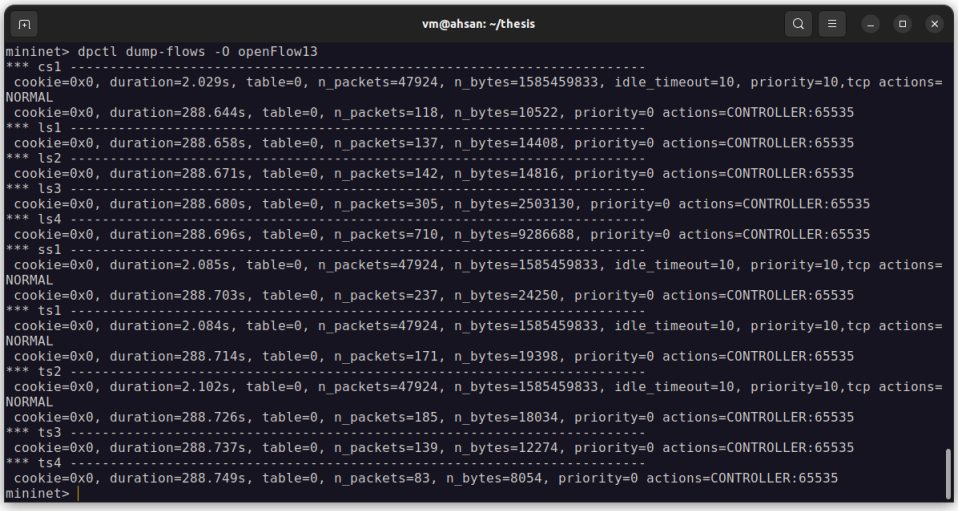


Figure 3.8: Flow Table Entries for the Combined Topology

3.6 Evaluation Metrics

Our SDN-based load-balancing methodology requires different Quality of Service (QoS) measuring tools and benchmarking instruments for performance evaluation.

3.6.1 Quality of Service Metrics

Network assessment depends heavily on the operation of Quality of Service mechanisms. The following key metrics are used for evaluation:

- **Round-Trip Time (RTT):** Measures the time it takes for a packet during its complete journey from its origin to its final destination. Network responsiveness directly depends on this parameter of quality assessment.
- **Latency:** Network packets meet delays because they have to pass through the network infrastructure. Lower latency indicates a more efficient network.
- **Throughput:** Network data transfer speed is measured in Mbps (UDP) or Gbps (TCP) to determine the success rate of data transmission.
- **Jitter:** The variation in packet arrival times. Excessive jitter in the network reduces the performance quality of systems running real-time functions.
- **Packet Loss:** The percentage of lost packets reveals the network reliability because it impacts how reliably data can be transmitted through the system.

3.6.2 Tools Used for the Performance Evaluation

We used some static traffic flows along with media traffic flows including large file transfer. To measure these metrics we employed the following network performance testing tools:

- **Ping:** ICMP echo requests enable the detection of RTT and packet loss through their transmission to target hosts.
- **iPerf:** This widely used tool provides functionalities for measuring network throughput, latency, and jitter tests. OSIRIS supports simultaneous testing of UDP protocol and TCP implementing traffic.
- **Wget:** Utilized for downloading files and measuring large transfer rates.
- **D-ITG (Distributed Internet Traffic Generator):** A powerful tool capable of generating realistic traffic patterns like VoIP, video streaming, Voice, DNS, PING, HTTP, etc to evaluate QoS parameters comprehensively.

Chapter 4

Testing and Result

This section contains the assessment and analysis of network performance output that emerges from two testing conditions: with load balancing and without load balancing. The evaluation of network performance conditions involved measuring different QoS elements such as RTT, throughput, latency, jitter, packet loss and bandwidth. The performance of these tests was done on four different client-server pairs. Here, the server was always tree-host-1 (*th1*), and the clients were star-host-2 (*sh1*), linear-host-1 (*lh1*), cluster-host-1 (*ch1*) and tree-host-4 (*th4*). In order to understand the behavior of the network comprehensively, both TCP and UDP traffic were considered during the experiments. The collected data was processed and analyzed using Matplotlib which plotted graphs for the key metrics shown in Figure 4.1.

```python
import matplotlib.pyplot as plt
import numpy as np

thp = np.array([500, 1000, 1500, 2000, 2500, 3000, 3500, 4000])
th4 = np.array([1.45, 1.70, 1.70, 2.25, 2.73, 3.10, 3.61, 4.13])
lh1 = np.array([0.982, 3.81, 1.28, 1.67, 2.07, 2.42, 2.81, 3.14])
ch1 = np.array([0.979, 1.24, 1.27, 1.69, 2.08, 2.37, 2.80, 3.15])
sh1 = np.array([1.08, 1.37, 1.38, 1.82, 2.20, 2.60, 3.01, 3.39])

plt.figure(figsize=(8, 5))

plt.plot(thp, th4, label='th1-th4', color='#955196', linestyle='-', linewidth=2, marker='o')
plt.plot(thp, lh1, label='th1-lh1', color='#58508d', linestyle='-', linewidth=2, marker='o')
plt.plot(thp, ch1, label='th1-ch1', color='#ef5675', linestyle='-', linewidth=2, marker='o')
plt.plot(thp, sh1, label='th1-sh1', color='#ffa600', linestyle='-', linewidth=2, marker='o')

plt.xlabel('Packet size')
plt.ylabel('Throughput(GBPS)')
plt.title('TCP Throughput for client-server pair')
plt.legend()

plt.grid(True)

plt.xticks(thp)

plt.yticks(np.arange(min(np.min(sh1), np.min(th4)), np.max(sh1)+0.5, 0.5))

plt.show()
```

Figure 4.1: Generating Graphs using Matplotlib

The results of each metric have been plotted to give clear visual insights into the performance of the network under SDN control. These tests yield valuable information on how the network performs under various types of load.

In our implementation, we have integrated a Machine Learning (ML) model into the SDN controller to classify network flows as either **mice flows** or **elephant flows**. This classification enables intelligent traffic management and improves overall

network efficiency. Based on some features the model dynamically categorizes the flow type and assists the controller in making optimal routing decisions. In the case of ***ping***, the controller was detecting the flow as **Mice Flow** as shown in Figure 4.2 and in the case of ***iperf*** or large media traffic show in Figure 4.3 the controller is successfully detecting the flow as a **Elephant Flow**.

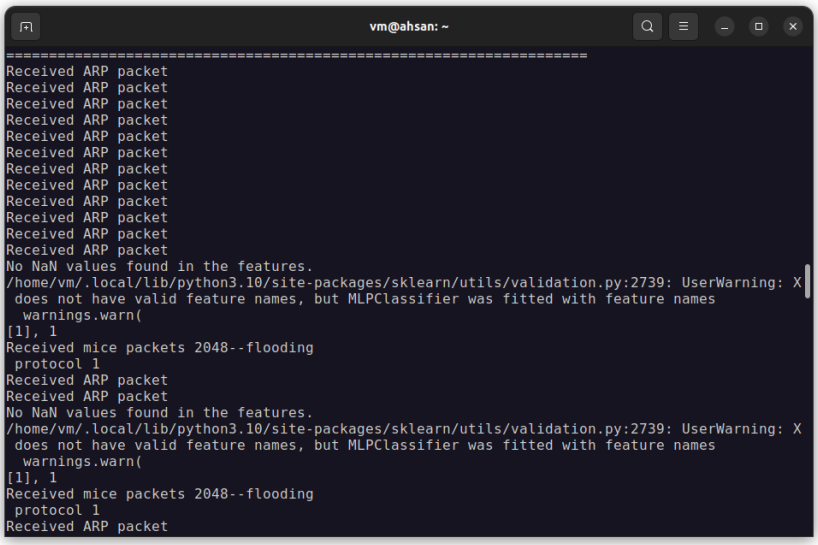


Figure 4.2: Detecting the flow as Mice

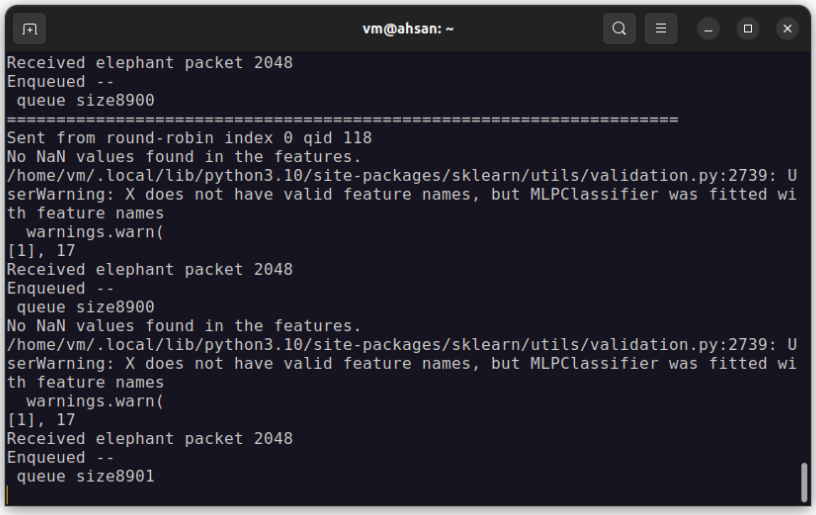


Figure 4.3: Detecting the flow as Elephant

The RYU controller implements distinct traffic handling methods after it recognizes the flow type. The controller sends Mice Flows first to paths with low congestion since Mice Flows need quick packet delivery. Elephant Flow generates network congestion if not properly managed so the controller uses the time domain set by our implementation as a strategy to spread traffic load for the prevention of network bottlenecks.

4.1 Without Load Balancing

This scenario examines network performance without a load-balancing mechanism where traffic follows a default static routing path potentially leading to congestion.

4.1.1 Throughput

To evaluate the network throughput without load balancing, ***iperf*** was used for TCP and UDP traffic tests shown in Figure 4.4. The throughput results for different packet sizes are analyzed below.

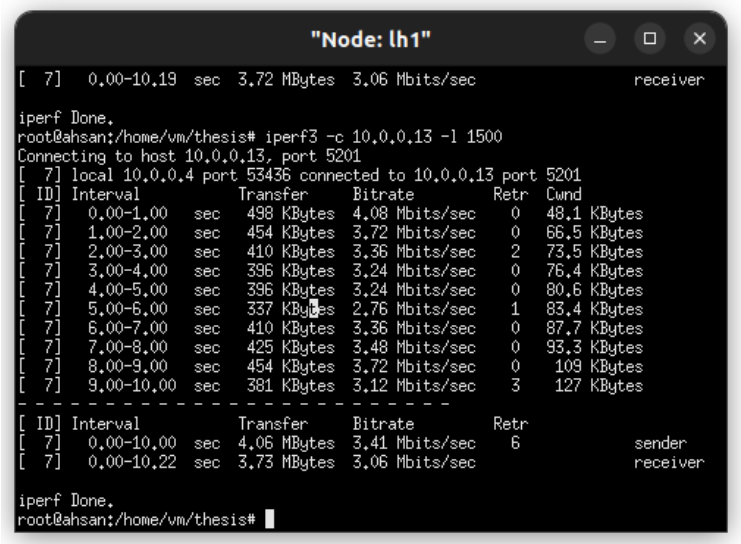


Figure 4.4: iperf Command for Sending Larger traffic

TCP Throughput: In Figure 4.5, the TCP throughput evaluation shows results for different client-server pairs without implementing the load balancer.

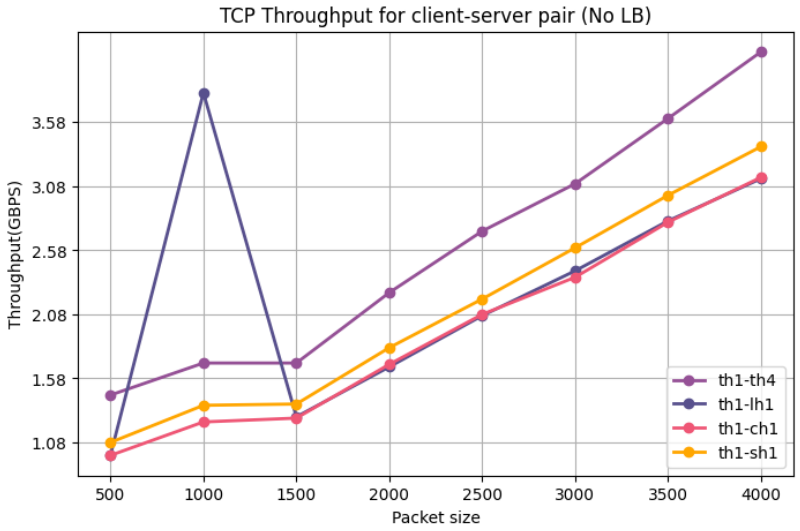


Figure 4.5: TCP Throughput for Different Packet Sizes (NoLB)

From the results:

- **th1-th4** shows an increasing trend, peaking at 4.13 Gbps for 4000-byte packets.
- **th1-lh1** has a fluctuating trend, reaching a peak of 3.14 Gbps at 4000 bytes.
- **th1-ch1** exhibits a gradual increase with a maximum throughput of 3.15 Gbps.
- **th1-sh1** has a steady increase, reaching 3.39 Gbps at 4000 bytes.

UDP Throughput: The UDP throughput values for different packet sizes are shown in Figure 4.6.

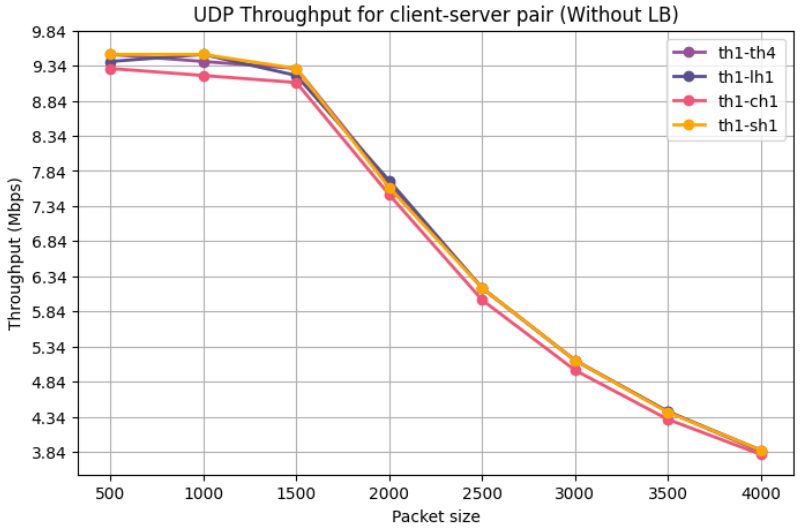


Figure 4.6: UDP Throughput for Different Packet Sizes (NoLB)

Our observations from the results:

- All pairs begin with a high throughput of around 9.5 Mbps, but performance declines as packet size increases.
- **th1-th4** drops to 3.84 Mbps, indicating a significant impact on large packet transmission.
- **th1-sh1** and **th1-lh1** also experience declines, stabilizing around 3.86 Mbps at 4000 bytes.
- **th1-ch1** shows the most significant reduction, ending at 3.80 Mbps, suggesting higher susceptibility to packet loss and congestion.

The results show that UDP throughput is more impacted by the lack of load balancing, likely due to its connectionless nature and lack of congestion control.

4.1.2 Jitter

Jitter measures the variation in packet delay, critical for real-time applications. The jitter results without load balancing are presented in Figure 4.7.

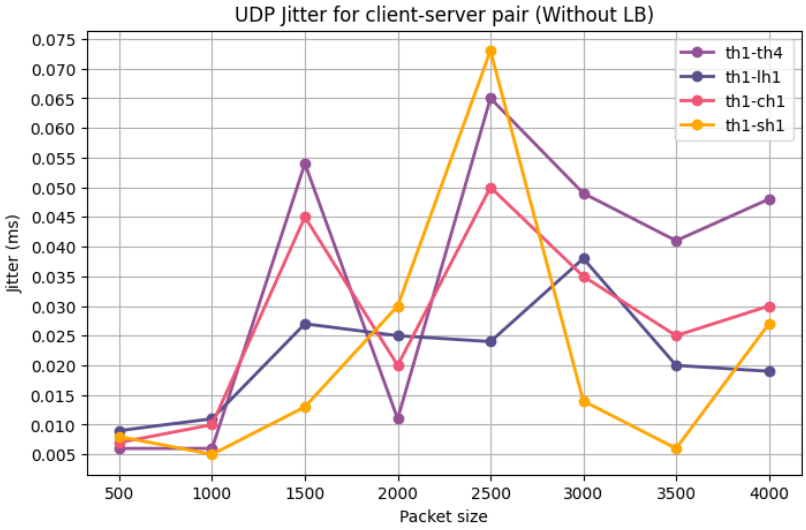


Figure 4.7: Jitter Evaluation for Different Packet Sizes (NoLB)

Our key observations from the results:

- **th1-th4**: Shows moderate jitter peaking at 0.065 ms, indicating slight instability in packet timing.
- **th1-sh1**: Peaks at 0.073 ms, which could impact voice and video quality in real-time applications.
- **th1-lh1**: Exhibits a consistently higher jitter, reaching 0.038 ms, reflecting less stable packet transmission.
- **th1-ch1**: Also shows variability, peaking at 0.050 ms, indicating sensitivity to network changes.

In the absence of load balancing, jitter values are notably higher, affecting the quality of real-time services.

4.1.3 Packet Loss

Packet loss is measured using *ping* and *iperf*. The packet loss percentages across varying packet sizes are illustrated in Figure 4.8.
Findings:

- **th1-th4**: Minor loss peaks at 1.3% at 1000 bytes, showing general stability.
- **th1-sh1**: Exhibits no packet loss, indicating reliable transmission on this path.
- **th1-lh1**: Encounters small losses, peaking at 0.37% at 1500 bytes, but remains stable otherwise.

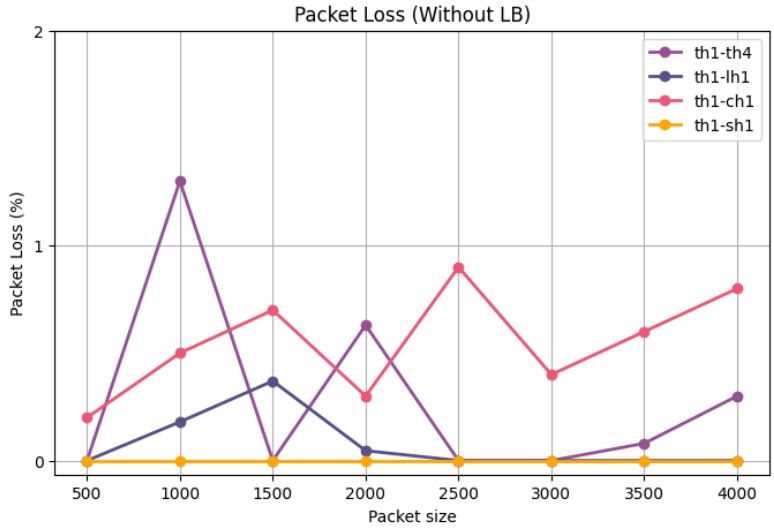


Figure 4.8: Packet Loss for Different Packet Sizes (NoLB)

- **th1-ch1**: Displays the highest packet loss, reaching up to 0.9%, reflecting potential congestion.

Overall, packet loss is low but more pronounced in certain pairs, highlighting the inefficiency of traffic distribution without load balancing.

4.1.4 Summary

load balancing.
Under no load balancing TCP throughput remains generally stable but is lower compared to load balancing scenarios. UDP throughput, on the other hand, degrades more quickly due to the lack of congestion management. Jitter values are higher, indicating less stable network conditions overall. While packet loss is generally low, it becomes more noticeable in paths like th1-ch1.
These findings emphasize the advantages of load balancing in improving network performance and minimizing delays.

4.2 Load Balancing

In this scenario, traffic is distributed based on the time domain set by us to manage the flow in a round robin scheduler fashion. The controller dynamically adjusts flow entries to optimize network utilization.

4.2.1 RTT and Latency

To measure RTT and latency we used the ***ping*** command between the designated server (*th1*) and each of the clients (*sh1*, *lh1*, *ch1*, *th4*) which is shown in the below Figure 4.9 and also Figure 4.10 is the recorded average RTT values.

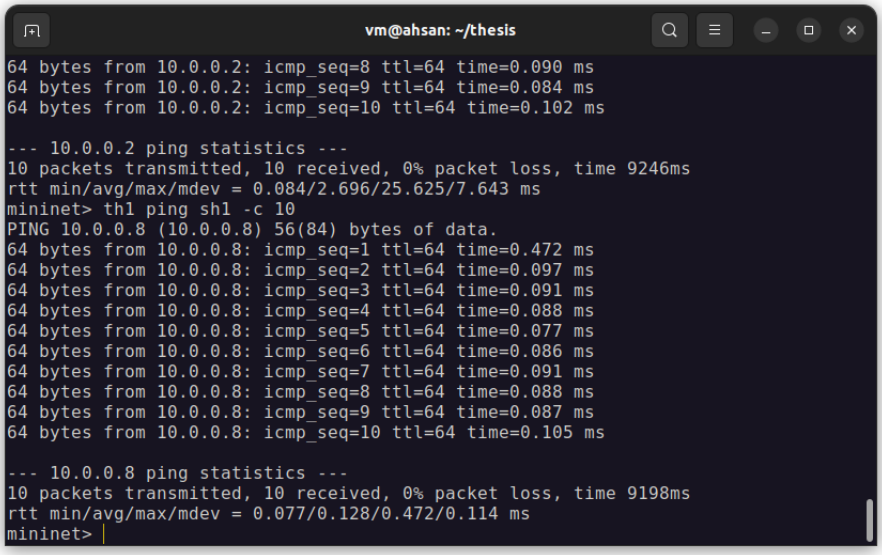


Figure 4.9: Running the ping Commands

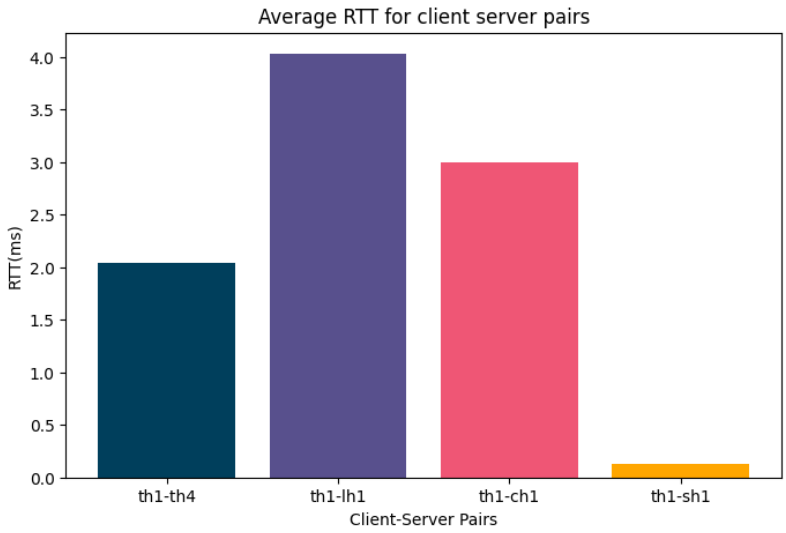


Figure 4.10: RTT values of the Four Client-Server Pairs (LB)

From the results, it is evident that the *th1-sh1* pair experiences the lowest RTT at 0.128 ms which indicates minimal delay due to closer proximity or fewer intermediate hops. The highest RTT is observed in the *th1-lh1* pair at 4.031 ms which may be attributed to increased routing complexity or additional network hops.

4.2.2 Throughput

To evaluate the network throughput under load balancing, we utilized ***iperf*** for TCP and UDP traffic tests. The throughput results for different packet sizes are analyzed below.

TCP Throughput: Figure 4.11 illustrates the TCP throughput for different client-server pairs.

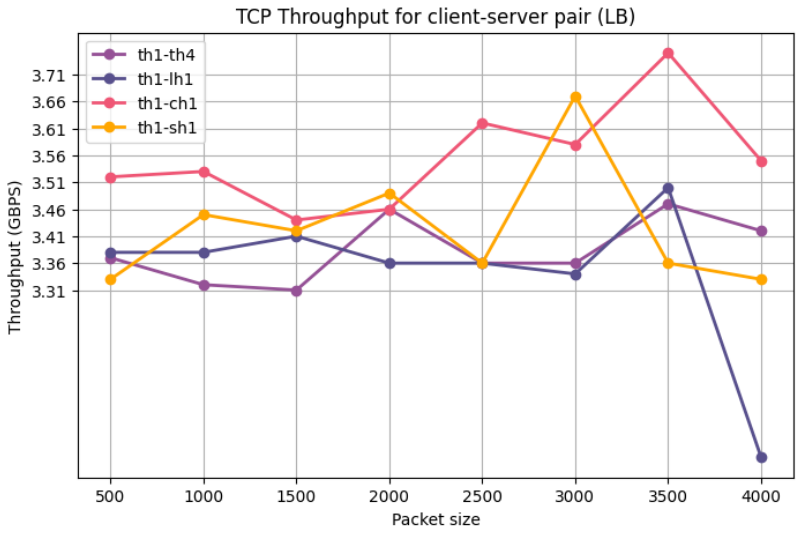


Figure 4.11: TCP Throughput for Different Packet Sizes (LB)

From the results:

- **th1-th4**: Throughput remains relatively stable, averaging around 3.42 Gbps, indicating minimal congestion because of load balancing.
- **th1-sh1**: Shows slight fluctuations with a peak of 3.67 Gbps at 3000 bytes, but overall remains consistent.
- **th1-lh1**: Experiences more variability, dropping to 3.00 Gbps at 4000 bytes, suggesting occasional congestion or inefficient routing.
- **th1-ch1**: Achieves the highest throughput of 3.75 Gbps, indicating that this path is less affected by traffic congestion.

These results suggest that TCP throughput is generally stable under load balancing, with better efficiency in larger packet sizes.

UDP Throughput: The UDP throughput values for different packet sizes are shown in Figure 4.12.
Our key observations from the results:

- All pairs initially start with a maximum throughput of 10.5 Mbps.
- **th1-th4** and **th1-ch1** gradually decrease, reaching 3.87 Mbps and 4.24 Mbps at 4000 bytes, respectively.

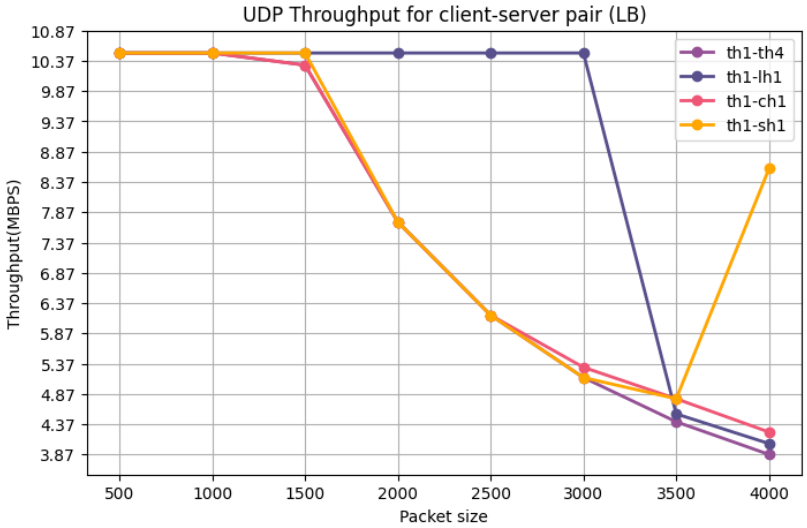


Figure 4.12: UDP Throughput for Different Packet Sizes (LB)

- **th1-lh1** maintains a stable 10.5 Mbps up to 3000 bytes but drops to 4.05 Mbps afterward.
- **th1-sh1** experiences fluctuations, peaking at 8.59 Mbps at 4000 bytes.

This demonstrates that load balancing generally maintains high UDP throughput, but performance varies with packet size due to network conditions.

4.2.3 Jitter

Jitter, the variation in packet delay, was analyzed using the latency values from ping tests. The jitter results are shown in Figure 4.13.

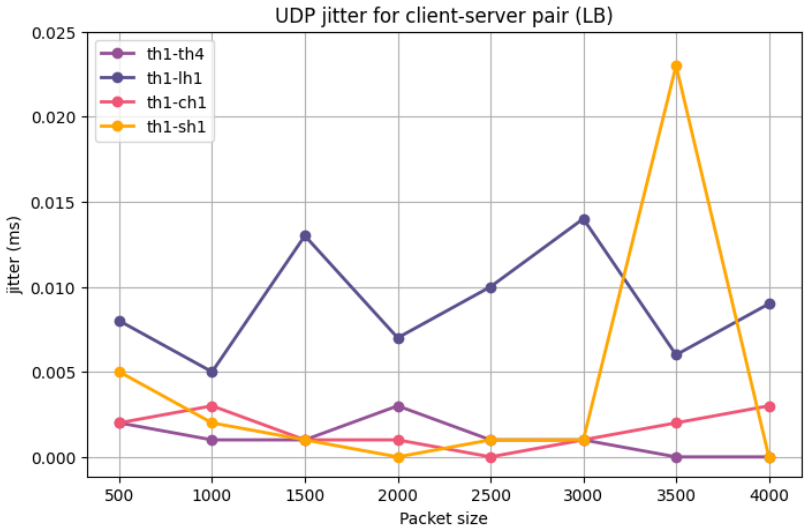


Figure 4.13: Jitter Evaluation for Different Packet Sizes (LB)

Key observations from the results:

- **th1-th4** maintains very low jitter (< 0.003 ms), ensuring stable transmission.

- **th1-lh1** has higher jitter, peaking at 0.014 ms.
- **th1-ch1** exhibits minimal jitter, with a slight peak at 0.003 ms.
- **th1-sh1** remains stable but shows a spike (0.023 ms) at 3500 bytes.

Low jitter values indicate good network stability under load balancing.

4.2.4 Packet Loss

Packet loss was measured using both ***ping*** and ***iperf***. Figure 4.14 displays the packet loss percentage across different packet sizes.

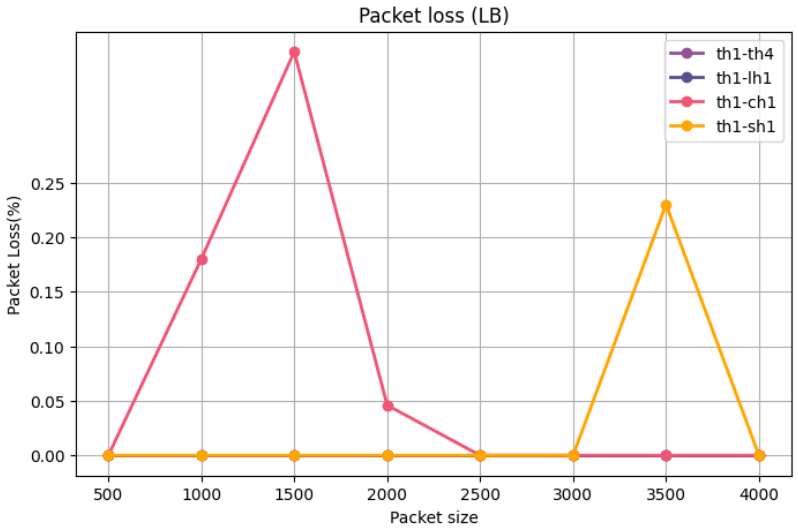


Figure 4.14: Packet Loss for Different Packet Sizes (LB)

From the Figure,

- **th1-th4** and **th1-lh1** have 0% packet loss across all packet sizes, indicating a stable connection.
- **th1-ch1** experiences minor loss at 1000-1500 bytes (0.18% and 0.37%), suggesting occasional network congestion.
- **th1-sh1** shows a small spike at 3500 bytes (0.23%), potentially due to traffic load variation.

These results suggest that load balancing minimizes packet loss in most cases, with only minor fluctuations under high traffic. Overall, the QoS metrics under load balancing show that RTT is low especially for *th1-sh1*, indicating efficient routing. TCP throughput increases with packet size, reaching up to 3.75 Gbps. UDP throughput remains high but gradually declines for larger packet sizes. Jitter remains minimal, showing stable transmission. Packet loss is negligible except for minor variations in *th1-ch1* and *th1-sh1*.

4.3 Load Balancing vs. Without Load Balancing

This section compares network performance in terms of throughput, latency, jitter, packet loss, D-ITG, and large video transfer under two scenarios: with load balancing and without load balancing.

4.3.1 Throughput Comparison

Throughput is a key performance indicator for network efficiency. The research evaluates throughput performance between TCP and UDP protocols when analyzing data transfer speed across client-server pairings under load balancing conditions. The figure at reference 4.15 demonstrates different throughputs achieved among four various client-server setups.

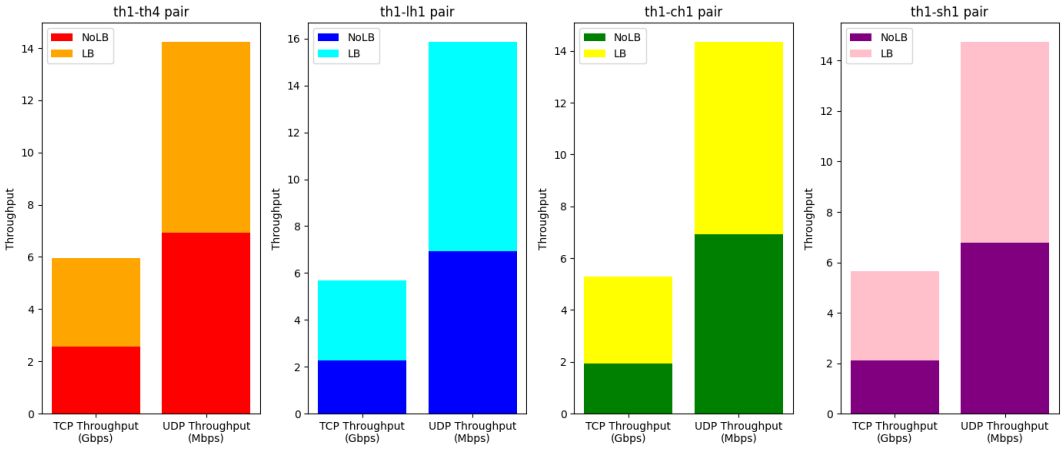


Figure 4.15: Throughput Comparison in 4 client-server pairs

TCP throughput reflects how well the network can handle reliable, connection-oriented traffic. Without load balancing, TCP throughput is 2.58 Gbps in *th1-th4* whereas with load balancing it improves to 3.38 Gbps. This suggests that traffic distribution leads to more efficient utilization of network resources. The throughput of *th1-lh1* increases from 2.272 Gbps (NoLB) to 3.426 Gbps (LB) which highlights that balanced routing reduces congestion and enhances data transmission efficiency. Overall, TCP throughput improves across all paths when load balancing is applied, reducing congestion and improving resource utilization. Besides TCP, UDP throughput in *th1-th4* also increases from 6.927 Mbps (NoLB) to 7.325 Mbps (LB) indicating that load balancing helps maintain more stable and higher performance. Also in *th1-lh1*, a small but notable improvement from **6.933 Mbps** (NoLB) to 8.948 Mbps (LB) suggests better packet handling under balanced conditions. Since UDP does not have built-in congestion control like TCP, load balancing prevents excessive packet loss improves delivery rates, and ensures a smoother experience for real-time applications.

4.3.2 Jitter and Packet Loss

Network performance indicators jitter and packet loss play a crucial role in monitoring systems using real-time applications including VoIP and video conferencing.

The Jitter and Packet Loss Comparison in an average between scenarios appears in Figure 4.16.

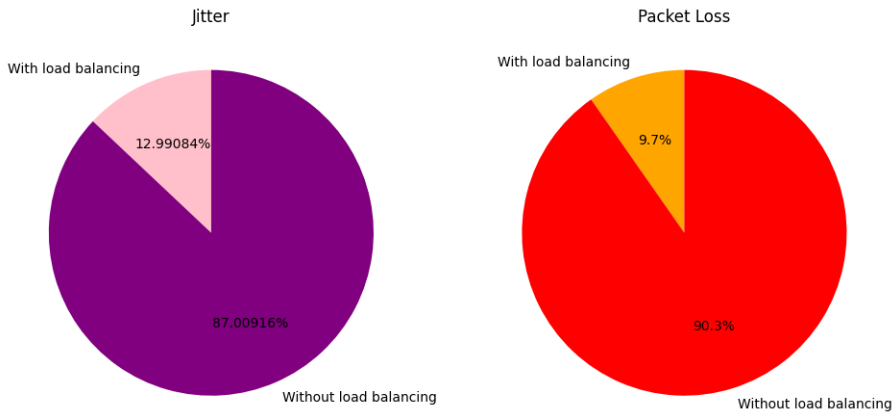


Figure 4.16: Jitter and Packet Loss Comparison

Jitter measures the variation in packet arrival time affecting real-time applications. Without load balancing, the jitter is significantly higher at 0.02659 compared to 0.00397 with load balancing. This reduction demonstrates that load balancing ensures consistent packet delivery, reducing the likelihood of voice or video disruptions. Packet loss occurs when packets fail to reach their destination, impacting data reliability. Without load balancing, packet loss is higher at 0.02581 whereas with load balancing it is reduced to 0.00278. The reduction suggests that load balancing optimizes packet routing minimizing packet drops and improving network reliability.

4.3.3 D-ITG Traffic Throughput

To evaluate real-world application performance, we analyzed various traffic types using the Distributed Internet Traffic Generator (D-ITG). We have used six different clients for sending different types of traffic to the th1 server. We used the following commands for generating the traffic,

Server Side:

- s1 → *ITGRecv*

Client Side:

- h1 → *ITGSend -a 10.0.0.13 -T TCP -c 1 -C 10 -t 30000*
- h2 → *ITGSend -a 10.0.0.13 -T UDP -c 1 -C 5 -t 20000 -p 53*
- h3 → *ITGSend -a 10.0.0.13 -T UDP -c 1 -C 1 -t 1000 -p 9*
- h4 → *ITGSend -a 10.0.0.13 -T UDP -c 10 -C 100 -t 20000*
- h5 → *ITGSend -a 10.0.0.13 -T UDP -c 20 -C 500 -t 60000*
- h6 → *ITGSend -a 10.0.0.13 -T TCP -c 10 -C 50 -t 120000*

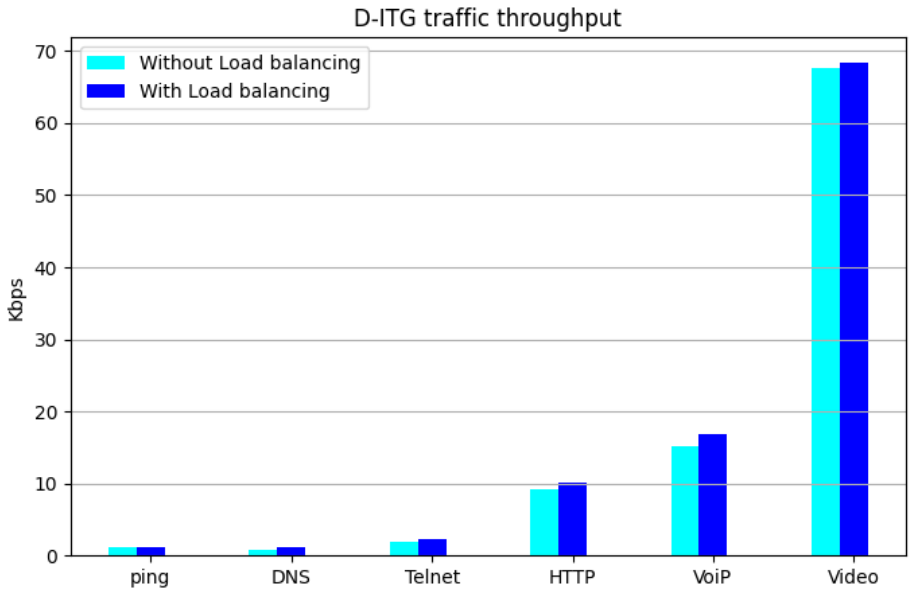


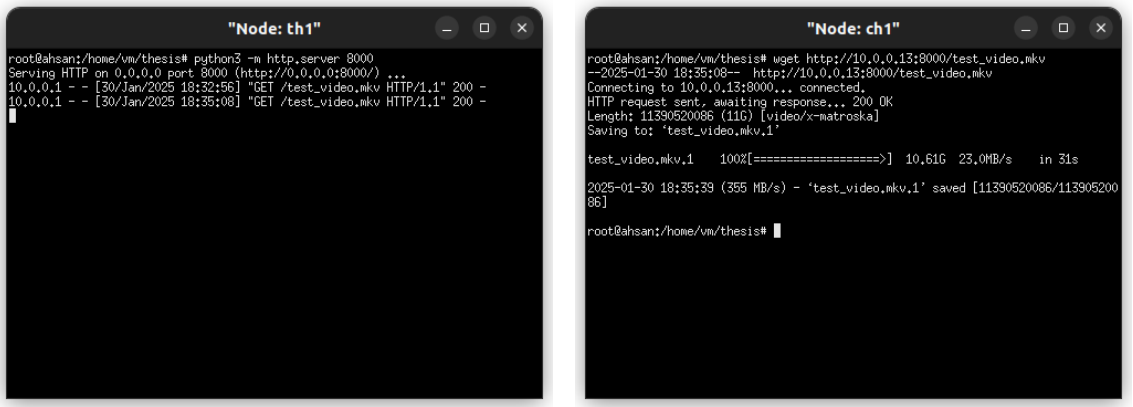
Figure 4.17: D-ITG Traffic Throughput Comparison

The results in Figure 4.17 highlight how load balancing affects different network services.

From the figure, there is minor improvements are observed with load balancing in Ping and DNS traffic. In the case of Telnet and HTTP, Load balancing increases the throughput. The most significant impact is seen in high-bandwidth applications, with video throughput improving from 67.57 Kbps to 68.45 Kbps.

4.3.4 Large Video Transfer Performance

Large file transfers test the efficiency of a network under high data loads. We sent an **11GB** Video file to a server from a client which is shown in Figure 4.18 and observed the result.



(a) Server Side (b) Client Side

Figure 4.18: Large Video Transfer using wget

We compare transfer speed and download time for a large video file which is shown in Figure 4.19.

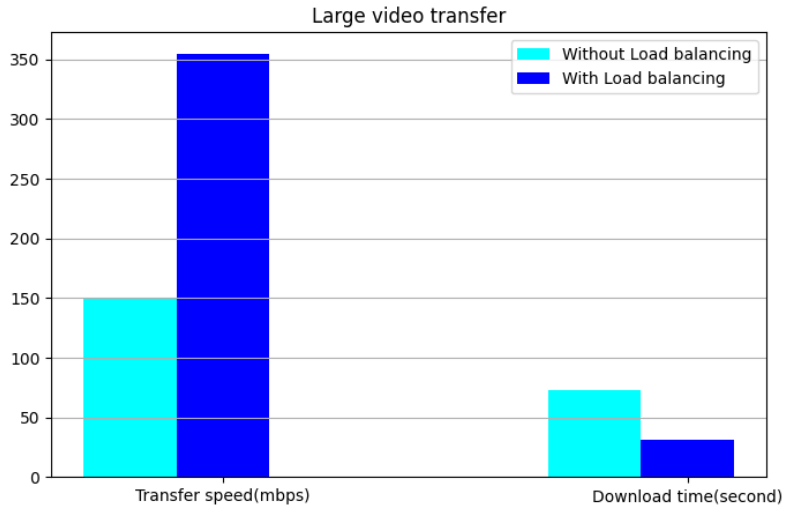


Figure 4.19: Large Video Transfer: Load Balancing vs. No Load Balancing

From the figure, without load balancing Transfer speed is limited to 150 Mbps, resulting in a download time of 73 seconds. On the other hand, with load balancing Transfer speed significantly increases to 355 Mbps, reducing the download time to 31 seconds. This substantial improvement demonstrates how load balancing efficiently utilizes multiple paths to optimize large file transfers.

Overall, these findings reinforce the importance of implementing load balancing in modern network infrastructures to achieve higher stability, efficiency, and reliability The table in 4.1 summarizes the average outcomes of the evaluation metrics including TCP and UDP throughput, and jitter in different client-server pairs in the custom topology in both scenarios.

Client-Server Pair	**TCP Throughput (*Gbps*) NoLB**	**TCP Throughput (*Gbps*) LB**	**UDP Throughput (*Mbps*) NoLB**	**UDP Throughput (*Mbps*) LB**	**Jitter (*ms*) NoLB**	**Jitter (*ms*) LB**
th1-th4	2.58	3.38	6.927	7.325	0.02659	0.00397
th1-lh1	2.272	3.426	6.933	8.948	0.02581	0.00278
th1-ch1	1.947	3.341	6.922	7.44	0.02436	0.00258
th1-sh1	2.106	3.556	6.774	7.9875	0.02390	0.00274

Table 4.1: Performance Comparison of SDN Load Balancing vs No Load Balancing

Chapter 5

Comparative Table

This section provides a comparative analysis of recent research covering various SDN aspects, performance, and load balancing. The following Table 5.1 provides a broad outline of the major contributions made in the field of SDN, technologies employed, algorithms developed by authors, and performance benchmarking in terms of jitter, RTT, latency, and throughput as given by various authors in SDN literature.

Author	**Research Focus**	**Technologies Used**	**Used Algo**	**Controller**	**Performance (Jitter, RTT, Latency, Throughput)**	**Used Topology**
Minardi et al. [61]	Joint VNE optimization with load balancing and data rate assignment in SDN.	SDN, SCA-VNE, SCA-R, Mininet, RYU.	Mi-ed Binary Linear Programming (MBLP), SCA-R.	RYU SDN controller.	Jitter: Reduced, RTT: Improved, Latency: Lowered, Throughput: Increased.	16-node SDN grid.
Cao and Rong [57]	TCP performance in SDN	Mininet, RYU, OpenvSwitch	iperf for bandwidth, GNUPLOT for visualization	RYU Controller	Jitter: Present, RTT: Increases with hops, Latency: Rises with more switches, Throughput: Decreases.	Multiple switches (up to 6), multiple hosts.

Binod et al. [64]	Traffic-driven load balancing in multi-controller SDN	Mininet, Ryu Controller, OpenFlow 1.3, D-ITG traffic generator	Decision Tree (CART), Logistic Regression, SVM, K-NN, XG-Boost, OP-TICS	Ryu Controller (Multi-Controller Setup)	Jitter: Reduced, Bit Rate: Increased, Packet Rate: Increased, Throughput: Improved	Multi-controller SDN environment with three controllers, hierarchical control plane
Wassie et al. [69]	QoS optimization in SDN using deep learning models for elephant flow detection and prediction	Mininet, Ryu Controller, Open vSwitch, Anaconda (TensorFlow, Keras), Iperf	Deep Neural Network (DNN), Convolutional Neural Network (CNN), Random Forest (RF), Naïve Bayes	Ryu Controller	Jitter: Not Mentioned, RTT: Not Mentioned, Latency: Reduced, Throughput: Improved (Measured UDP throughput: 9.85-9.99 Mbits/sec)	Custom linear topology with multiple switches and hosts (Mininet-based simulation)
Ram et al.[62]	Performance of POX vs. Ryu in wired/wireless networks.	Mininet-Wi-Fi, D-ITG.	Ryu, POX.	Ryu outperforms POX.	Jitter: Not Present, RTT: Not Present, Latency: Ryu better, Throughput: Ryu better.	Single, linear, tree.
Vijay et al.[67]	SDN in large IoT networks with hierarchical tree topology.	Mininet, iperf.	-	-	Jitter: Not Present, RTT: Not Present, Latency: Minimal RTT, Throughput: Not Present.	Hierarchical tree.

Gonzalez et al. [43]	ML-based anomaly detection in SDN for embedded systems	Mininet, Open vSwitch, QEMU, TinyCore Linux, Scikit-learn	DT, SVM, NN	OpenDaylight, RYU, ONOS	Jitter: Not Mentioned, RTT: Not Mentioned, Latency: Reaction time varies (DT: 3.17-3.87 ms, SVM: 3.55-4.29 ms, NN: 8.99-11.24 ms), Throughput: Not Mentioned	Experimental SDN architecture with embedded systems and anomaly detection module
Girish et al.[59]	SDN-based cloud computing load balancing.	-	PSO, GWO, Link Load Evaluation.	-	Jitter: Not Present, RTT: Not Present, Latency: Lower, Throughput: Increased.	-
Ye et al.[70]	ILBPS for SDN load-balancing and path selection.	Mininet, Ryu.	SSA, GCBAC, CNN, ABC-SP.	Ryu.	Jitter: Reduced 23.32%–36.67%, RTT: Not Present, Latency: Not Present, Throughput: Improved 10.75%–30.71%.	GÉANT network.
Yzzogh et al.[55]	SDN-based load balancing in cloud computing.	SDN, Ryu, OpenFlow.	Dynamic traffic engineering, multi-path routing.	SDN (Ryu).	Jitter: Minimized, RTT: Reduced, Latency: Lowered, Throughput: Enhanced.	Cloud data centers.
Mokoena et al.[49]	SDN with OpenFlow protocol for network management.	GNS3, Oracle VMs, Mininet, Cisco switches, routers, Ansible, Git.	SDN + OpenFlow.	-	Jitter: Not Present, RTT: Not Present, Latency: Not Present, Throughput: Not Present.	Centralized network architecture.

Osei et al.[51]	Load balancing in SDN (traditional vs AI techniques).	-	HDW (Hash IP, Weighted Scheduler, Dynamic Routing).	-	Jitter: Not Present, RTT: Not Present, Latency: Not Present, Throughput: 15% improvement with HDW.	-
Lokuliyana et al.[47]	QoS evaluation of SDN controllers (POX, RYU).	Mininet, Miniedit, OpenFlow.	POX, RYU, hybrid framework.	POX, RYU.	Jitter: POX 38.33% lower, RTT: Not Present, Latency: POX 225.725% lower, Throughput: RYU 13.40% higher.	Linear, Mesh, Star, Tree topology.
Ajay et al. [33]	Priority-based load balancing in cloud computing	CloudSim 3.0.3, Virtual Machines	Priority Weighted Round Robin, Weighted Round Robin, Dynamic Weighted Round Robin, Round Robin	CloudSim Scheduler	Execution Cost: Reduced, Total Completion Time: Improved, Waiting Time: Decreased, Turnaround Time: Improved, Higher priority tasks completed faster	50 Virtual Machines (VMs) with heterogeneous cloudlets
Hasan et al.[11]	Benchmarking SDN performance using Mininet and Floodlight controller.	Mininet, Floodlight controller.	-	Floodlight.	Jitter: Not Present, RTT: Not Present, Latency: Not Present, Throughput: Monitored.	-

Yassin et al. [21]	Load balancing in cloud computing using SDN	OpenStack, HAProxy, Apache2, SDN	Round Robin	SDN Controller in OpenStack	Jitter: Not Present, RTT: Not Present, Latency: Improved, Throughput: Increased	Two clients, one server (before & after load balancing)
Ejaz et al.[23]	SDN load balancing with DALB, SDN-NFV, vSDN controllers.	-	DALB, SDN-NFV, vSDN.	vSDN.	Jitter: Not Present, RTT: Not Present, Latency: 41% less delay, Throughput: Improved.	-
Eissa et al.[6]	SDN architecture and OpenFlow protocol.	Mininet, Ryu controller, Pyretic, Python, Frenetic.	-	Ryu.	Jitter: Not Present, RTT: Not Present, Latency: Not Present, Throughput: Not Present.	-
Islam et al.[9]	Evaluation of SDN controllers in wireless networks.	Mininet, iperf.	Round-robin, Shortest Path, Load Balancing.	Ryu, POX, ONOS, Floodlight.	Jitter: Not Present, RTT: Not Present, Latency: Not Present, Throughput: Monitored using iperf.	Wireless network emulation.
Eissa et al.[1]	Improving network management with SDN and Procera.	SDN, OpenFlow, FRP, Procera.	FRP, OpenFlow	Procera, NO-, Floodlight, Maestro.	Jitter: Not Present, RTT: Not Present, Latency: Improved, Throughput: Not Present.	Campus and home networks.

Table 5.1: Comparative Review of SDN-related Research Papers

Chapter 6

Discussion

Our approach effectively differentiates between elephant flows and mice flows using the load balancer algorithm in the RYU controller, enabling more customized traffic management strategies. This capability is particularly advantageous in hybrid network environments where traffic characteristics vary significantly. Additionally, our analysis of TCP and UDP traffic behavior across different scenarios provides valuable insights into SDN's ability to accommodate diverse network applications.

However, our research also identifies certain limitations. The effectiveness of SDN-based load balancing largely depends on selecting the right controller and optimizing flow management algorithms. While our simulations yielded promising results, real-world deployments may introduce additional challenges, such as hardware limitations and security vulnerabilities. Moreover, SDN's centralized control architecture while beneficial for network management raises concerns about single points of failure and scalability in large-scale network environments.

Chapter 7

Conclusion and Future Work

This study critically analyzes the role of Software Defined Networking (SDN) in load balancing by utilizing the RYU controller and machine learning-based flow classification. The findings confirm that our proposed load balancer algorithm in the RYU controller along with SDN can significantly enhance network performance by dynamically managing traffic distribution, leading to higher throughput and reduced latency and packet loss. By addressing the inefficiencies of traditional load-balancing methods, our proposed algorithm emerges as a viable solution for next-generation network infrastructures. However, deploying SDN in large-scale environments requires careful consideration of factors such as security, scalability, and compatibility with existing network protocols.

Our implementation, which simulates a hybrid SDN topology using OpenFlow-enabled switches in Mininet, demonstrates the effectiveness of SDN-based load balancing. Key performance metrics including throughput, latency, jitter, and packet loss were analyzed for TCP and UDP traffic between multiple client-server pairs. The results showed variations with higher TCP throughput for certain pairs and lower performance for more complex network paths. Although overall packet losses were minimal, minor losses in congested network scenarios highlight the need for more advanced traffic optimization techniques.

Future research can explore intelligent load balancing strategies that integrate real-world classification and predictive traffic management. The deployment of distributed SDN controllers could help address single point failure and scalability challenges making SDN more practical for large-scale networks. Furthermore, security remains a major concern and future advances may focus on AI-driven anomaly detection security systems frameworks to protect SDN networks from cyber threats. Real-world testing in large-scale environments is essential to validate the real-world effectiveness of SDN-based load balancing.

Bibliography

[1] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[2] A. De Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," in *2014 27th biennial symposium on communications (QBSC)*, IEEE, 2014, pp. 71–75.

[3] I. H. Abdulqadder, D. Zou, I. T. Aziz, B. Yuan, and W. Dai, "Deployment of robust security scheme in sdn based 5g network over nfv enabled cloud environment," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 2, pp. 866–877, 2018.

[4] T. Bakhshi, "Securing wireless software defined networks: Appraising threats, defenses & research challenges," in *2018 International Conference on Advancements in Computational Sciences (ICACS)*, IEEE, 2018, pp. 1–6.

[5] A. K. Arahunashi, S. Neethu, and H. R. Aradhya, "Performance analysis of various sdn controllers in mininet emulator," in *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, IEEE, 2019, pp. 752–756.

[6] H. A. Eissa, K. A. Bozed, and H. Younis, "Software defined networking," in *2019 19th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, IEEE, 2019, pp. 620–625.

[7] S. Ejaz, Z. Iqbal, P. A. Shah, B. H. Bukhari, A. Ali, and F. Aadil, "Traffic load balancing using software defined networking (sdn) controller as virtualized network function," *IEEE Access*, vol. 7, pp. 46 646–46 658, 2019.

[8] M. Iqbal, F. Iqbal, F. Mohsin, M. Rizwan, and F. Ahmad, "Security issues in software defined networking (sdn): Risks, challenges and potential solutions," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 10, 2019.

[9] S. Islam, M. A. I. Khan, S. T. Shorno, S. Sarker, and M. A. Siddik, "Performance evaluation of sdn controllers in wireless network," in *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, IEEE, 2019, pp. 1–5.

[10] Y. Liu, Z. Zeng, X. Liu, X. Zhu, and M. Z. A. Bhuiyan, "A novel load balancing and low response delay framework for edge-cloud network based on sdn," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5922–5933, 2019.

[11] M. Hasan, H. Dahshan, E. Abdelwanees, and A. Elmoghazy, "Sdn mininet emulator benchmarking and result analysis," in *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, IEEE, 2020, pp. 355–360.

[12] H. M. Noman and M. N. Jasim, "Pox controller and open flow performance evaluation in software defined networks (sdn) using mininet emulator," in *IOP conference series: materials science and engineering*, IOP Publishing, vol. 881, 2020, p. 012 102.

[13] A. Shaghaghi, M. A. Kaafar, R. Buyya, and S. Jha, "Software-defined network (sdn) data plane security: Issues, solutions, and future directions," *Handbook of Computer Networks and Cyber Security: Principles and Paradigms*, pp. 341–387, 2020.

[14] S. Yang, L. Cui, Z. Chen, and W. Xiao, "An efficient approach to robust sdn controller placement for security," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1669–1682, 2020.

[15] S. Ahmad and A. H. Mir, "Scalability, consistency, reliability and security in sdn controllers: A survey of diverse sdn controllers," *Journal of Network and Systems Management*, vol. 29, pp. 1–59, 2021.

[16] M. Alotaibi and A. Nayak, "Linking handover delay to load balancing in sdn-based heterogeneous networks," *Computer Communications*, vol. 173, pp. 170–182, 2021.

[17] B. Bwalya and A. Zimba, "An sdn approach to mitigating network management challenges in traditional networks," *International Journal on Information Technologies and Security*, vol. 13, no. 4, pp. 3–14, 2021.

[18] D. Cabarkapa and D. Rancic, "Performance analysis of ryu-pox controller in different tree-based sdn topologies.," *Advances in Electrical & Computer Engineering*, vol. 21, no. 3, 2021.

[19] D. Chasaki and C. Mansour, "Sdn security through system call learning," in *2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, IEEE, 2021, pp. 1–6.

[20] M. H. H. Khairi, S. H. S. Ariffin, N. M. A. Latiff, *et al.*, "Detection and classification of conflict flows in sdn using machine learning algorithms," *IEEE Access*, vol. 9, pp. 76 024–76 037, 2021.

[21] S. Kumar and A. Dumka, "Load balancing with the help of round robin and shortest job first scheduling algorithm in cloud computing," in *Proceedings of International Conference on Machine Intelligence and Data Science Applications: MIDAS 2020*, Springer, 2021, pp. 213–223.

[22] D. Lunagariya and B. Goswami, "A comparative performance analysis of stellar sdn controllers using emulators," in *2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, IEEE, 2021, pp. 1–9.

[23] Y. A. H. Omer, M. A. Mohammedel-Amin, and A. B. A. Mustafa, "Load balance in cloud computing using software defined networking," in *2020 International conference on computer, control, electrical, and electronics engineering (ICCCEEE)*, IEEE, 2021, pp. 1–6.

[24] M. Rajesh *et al.*, "Study on sdn with security issues using mininet," *Recent Trends in Intensive Computing*, vol. 39, p. 104, 2021.

[25] Z. I. Rasool, R. S. Abd Ali, and M. M. Abdulzahra, "Network management in software-defined network: A survey," in *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, vol. 1094, 2021, p. 012 055.

[26] J. Sherwin and C. J. Sreenan, "Software-defined networking for data centre network management: A survey," *arXiv preprint arXiv:2106.10014*, 2021.

[27] N. M. Yungaicela-Naula, C. Vargas-Rosales, and J. A. Perez-Diaz, "Sdn-based architecture for transport and application layer ddos attack detection by using machine and deep learning," *IEEE Access*, vol. 9, pp. 108 495–108 512, 2021.

[28] O. Yurekten and M. Demirci, "Sdn-based cyber defense: A survey," *Future Generation Computer Systems*, vol. 115, pp. 126–149, 2021.

[29] S. Abuthahir, S. C. B. Jaganathan, R. Saha, *et al.*, "An efficient enhanced dynamic load balancing weighted round robin algorithm for virtual machine in cloud computing," *Journal of Algebraic Statistics*, vol. 13, no. 2, pp. 2121–2128, 2022.

[30] O. M. Alssaheli, Z. Z. Abidin, N. Zakaria, and Z. A. Abas, "Software defined network based load balancing for network performance evaluation," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 4, 2022.

[31] N. S. Bülbül, D. Ergenç, and M. Fischer, "Towards sdn-based dynamic path reconfiguration for time sensitive networking," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2022, pp. 1–9.

[32] F. Chahlaoui, H. Dahmouni, and H. El Alami, "Multipath-routing based load-balancing in sdn networks," in *2022 5th Conference on Cloud and Internet of Things (CIoT)*, IEEE, 2022, pp. 180–185.

[33] A. Katangur, S. Akkaladevi, and S. Vivekanandhan, "Priority weighted round robin algorithm for load balancing in the cloud," in *2022 IEEE 7th international conference on smart cloud (SmartCloud)*, IEEE, 2022, pp. 230–235.

[34] Y. Liu, H. Gu, Z. Zhou, and N. Wang, "Rslb: Robust and scalable load balancing in software-defined data center networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4706–4720, 2022.

[35] J. Ma, R. Jin, L. Dong, G. Zhu, and X. Jiang, "Implementation of sdn traffic monitoring based on ryu controller," in *international symposium on computer applications and information systems (ISCAIS 2022)*, SPIE, vol. 12250, 2022, pp. 203–212.

[36] S. P. Praveen, P. Sarala, T. K. M. Kumar, S. G. Manuri, V. S. Srinivas, and D. Swapna, "An adaptive load balancing technique for multi sdn controllers," in *2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS)*, IEEE, 2022, pp. 1403–1409.

[37] Y. Wang, R. Liu, Y. Li, Z. Chen, N. Zhang, and B. Han, "Sdn controller network load balancing approach for cloud computing data center," in *2022 14th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, IEEE, 2022, pp. 242–246.

[38] F. A. Yaseen, N. A. Alkhalidi, and H. S. Al-Raweshidy, "She networks: Security, health, and emergency networks traffic priority management based on ml and sdn," *IEEE Access*, vol. 10, pp. 92 249–92 258, 2022.

[39] M. Ali, A. I. Jehangiri, O. I. Alramli, *et al.*, "Performance and scalability analysis of sdn-based large-scale wi-fi networks," *Applied Sciences*, vol. 13, no. 7, p. 4170, 2023.

[40] S. Avinash, A. S. Srikar, V. P. Naik, and S. Bhaskaran, "Sdn-based hybrid load balancing algorithm," in *2023 3rd International Conference on Intelligent Technologies (CONIT)*, IEEE, 2023, pp. 1–5.

[41] S. Bhardwaj and A. Girdhar, "Network traffic analysis in software-defined networking using ryu controller," *Wireless Personal Communications*, vol. 132, no. 3, pp. 1797–1818, 2023.

[42] J. Gómez, V. H. Riaño, and G. Ramirez-Gonzalez, "Traffic classification in ip networks through machine learning techniques in final systems," *IEEE Access*, vol. 11, pp. 44 932–44 940, 2023.

[43] C. Gonzalez and S. M. Charfadine, "Sdn controllers and ml-based anomaly detection in embedded systems: A comparative analysis," in *2023 10th International Conference on Wireless Networks and Mobile Communications (WINCOM)*, IEEE, 2023, pp. 1–6.

[44] G. Karn, B. Sapkota, and B. R. Dawadi, "Traffic classification and load balancing in sdn environment," 2023.

[45] D. A. Khoa, N. T. Kiem, N. T. Kien, N. N. Tuan, and N. H. Thanh, "Traffic-adaptive scheme for sdn control plane with containerized architecture," in *2023 17th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, IEEE, 2023, pp. 1–6.

[46] P. Krishnan, K. Jain, A. Aldweesh, P. Prabu, and R. Buyya, "Openstackdp: A scalable network security framework for sdn-based openstack cloud infrastructure," *Journal of Cloud Computing*, vol. 12, no. 1, p. 26, 2023.

[47] S. Lokuliyana, P. Wijesiri, A. Jayakody, and P. Abeyagunawardhena, "Qos evaluation of sdn controllers: Pox and ryu," in *2023 5th International Conference on Advancements in Computing (ICAC)*, IEEE, 2023, pp. 709–714.

[48] Y. Maleh, Y. Qasmaoui, K. El Gholami, Y. Sadqi, and S. Mounir, "A comprehensive survey on sdn security: Threats, mitigations, and future directions," *Journal of Reliable Intelligent Environments*, vol. 9, no. 2, pp. 201–239, 2023.

[49] K. M. R. Mokoena, R. L. Moila, and M. Velempini, "Improving network management with software defined networking using openflow protocol," in *2023 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, IEEE, 2023, pp. 1–5.

[50] D. Nuñez-Agurto, W. Fuertes, L. Marrone, E. Benavides-Astudillo, and M. Vásquez-Bermúdez, "Traffic classification in software-defined networking by employing deep learning techniques: A systematic literature review," in *International Conference on Technologies and Innovation*, Springer, 2023, pp. 67–80.

[51] E. Osei Kofi and E. Ahene, "Enhanced network load balancing technique for efficient performance in software defined network," *Plos one*, vol. 18, no. 4, e0284176, 2023.

[52] H. M. U. Rahman, G. Bahoo, L. Zafar, I. Al-Oqily, H. A. Khattak, and A. Ahmad, "Empirical performance evaluation of open source sdn controllers in different network topologies," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2023, pp. 1–6.

[53] M. C. Saxena, M. Sabharwal, and P. Bajaj, "Review of sdn-based load-balancing methods, issues, challenges, and roadmap," *International journal of electrical and computer engineering systems*, vol. 14, no. 9, pp. 1031–1049, 2023.

[54] K. Vani and K. RamaMohanBabu, "An intelligent server load balancing based on multi-criteria decision-making in sdn," *International journal of electrical and computer engineering systems*, vol. 14, no. 4, pp. 433–442, 2023.

[55] H. Yzzogh and H. Benaboud, "Using sdn to enhance load balancing in cloud computing: An overview and future directions," in *2023 6th International Conference on Advanced Communication Technologies and Networking (CommNet)*, IEEE, 2023, pp. 1–6.

[56] S. Bhaskaran and M. Supriya, "Study on networking models of sdn using mininet and miniedit," in *2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*, IEEE, 2024, pp. 1159–1164.

[57] R. Cao, "Tcp performance analysis for software-defined networks," in *2024 4th International Symposium on Computer Technology and Information Science (ISCTIS)*, IEEE, 2024, pp. 65–69.

[58] Ö. Çubukçu, A. Çubukçu, A. Kavak, and K. Küçük, "Sdn and nfv based dynamic qos management for 5g and beyond networks," in *2024 32nd Signal Processing and Communications Applications Conference (SIU)*, IEEE, 2024, pp. 1–4.

[59] B. Girish, B. Subramanaya, S. Rohan, and V. N. Priya, "Optimizing data center load balancing in cloud computing with sdn controller," in *2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS)*, IEEE, vol. 1, 2024, pp. 1–6.

[60] S. Kaczmarek and J. A. Litka, "Impact of sdn controller's performance on quality of service," *IEEE Access*, 2024.

[61] M. Minardi, T. X. Vu, I. Maity, C. Politis, and S. Chatzinotas, "Traffic-aware virtual network embedding with joint load balancing and datarate assignment for sdn-based networks," *IEEE Transactions on Network and Service Management*, 2024.

[62] A. Ram and S. K. Chakraborty, "Analysis of software-defined networking (sdn) performance in wired and wireless networks across various topologies, including single, linear, and tree structures," *Indian Journal of Information Sources and Services*, vol. 14, no. 1, pp. 39–50, 2024.

[63] A. O. Salau and M. M. Beyene, "Software defined networking based network traffic classification using machine learning techniques," *Scientific Reports*, vol. 14, no. 1, p. 20 060, 2024.

[64] B. Sapkota, B. R. Dawadi, S. R. Joshi, and G. Karn, "Traffic-driven controller-load-balancing over multi-controller software-defined networking environment," *Network*, vol. 4, no. 4, pp. 523–544, 2024.

[65] R. C. Tanguturi, J. Amutharaj, N. Prakash, K. Subramani, S. N. Taqui, *et al.*, "Analysis of the performance of software defined networks (sdn) versus networks with tcp/ip architecture," in *2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*, IEEE, 2024, pp. 342–350.

[66] T. Ťaptík, M. Hraška, J. Papán, and M. Janovec, "Comparison of basic sdn controllers," in *2024 34th International Conference Radioelektronika (RADIOELEKTRONIKA)*, 2024, pp. 1–4. DOI: 10.1109/RADIOELEKTRONIKA61599.2024.10524054.

[67] S. Vijay and M. Banga, "Managing large iot network using sdn and hierarchical tree topology," *Educational Administration: Theory and Practice*, vol. 30, no. 5, pp. 6484–6495, 2024.

[68] G. Wassie, *Sdn dataset*, https://github.com/getahunwassie/SDN_Dataset, 2024.

[69] G. Wassie, J. Ding, and Y. Wondie, "Detecting and predicting models for qos optimization in sdn," *Journal of Computer Networks and Communications*, vol. 2024, no. 1, p. 3 073 388, 2024.

[70] Z. Ye, G. Sun, and M. Guizani, "Ilbps: An integrated optimization approach based on adaptive load-balancing and heuristic path selection in sdn," *IEEE Internet of Things Journal*, vol. 11, no. 4, pp. 6144–6157, 2024.