Republic of Iraq
Ministry of Higher Education and Scientific Research
University of Babylon
College of Information Technology
Information Network Department

An Intelligent Dynamic Load Balancing for Improving QoS and Workload Distribution in Software Defined Networks

A Dissertation
Submitted to the Council of the College of Information Technology at University of Babylon in Partial Fulfillment of the Requirements for the Degree of Doctorate of Philosophy in Information Technology/ Information Network

By
Maghrib Abidalreda Maky Abidalreda

Supervised by
Prof. Dr. Wesam S. Bhaya

2023 A.D. **1445 A.H.**

بِسْمِ اللَّـهِ الرَّحْمَـٰنِ الرَّحِيمِ

(يَرْفَعِ اللَّهُ الَّذِينَ آمَنُوا مِنكُمْ وَالَّذِينَ أُوتُوا الْعِلْمَ دَرَجَاتٍ وَاللَّهُ بِمَا تَعْمَلُونَ خَبِيرٌ)

صدقَ اللهُ العليُّ العظيم

سورة المجادلة ، آية : 11

Supervisor Certification

I certify that this dissertation was prepared under my supervision at the Department of Information Network / Collage of Information Technology / Babylon University, by **Maghrib Abidalreda Maky Alramahi** as a partial fulfillment of the requirements for the degree of **Ph.D. in Information Technology.**

Signature:
Name: Dr. Wesam S. Bhaya
Title: Professor
Date: / / 2023

The Head of the Department Certification

In view of the available recommendation, we forward this dissertation for debate by the examining committee.

Signature:
Asst. Prof. Dr. Alharith A. Abdullah
Head of Information Networks Department
Date: / / 2023

Declaration

I hereby declare that this dissertation, **An Intelligent Dynamic Load Balancing for Improving QoS and Workload Distribution in Software Defined Networks** submitted to University of Babylon in partial fulfillment of requirements for the degree of Doctorate of Philosophy in Information Technology-Information Network has not been submitted as an exercise for a similar degree at any other University. I also certify that this work described here is entirely my own except for reports and summaries whose sources are appropriately cited in the references

Signature:

Name: **Maghrib Abidalreda Maky Alramahi**

Date: /   / 2023

Acknowledgments

All praise be to ALLAH Almighty who helped me complete this task successfully and my utmost respect to his last Prophet Mohammad PBUH and to the hero of Islam born in the heart of Kaaba Amir-Al-Mo’mineen, Al-Imam Ali Ibn Abi Talib.

First, I express my deep appreciation to my supervisor, **Prof. Dr. Wesam S. Bhaya**, for his invaluable guidance, supervision, and untiring efforts during the course of this work.

I express my profound gratitude to my mother, wife, and children for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this dissertation. I want to declare my deep thanks and appreciation to **my brothers, sisters, and friends** for their patience, encouragement, and help during the work.

Thanks and appreciation to all **my teachers and the staff at the College of Information Technology** for their great assistance during my studies.

Last but not least, I would like to thank all the kind, helpful, and lovely people who helped me directly or indirectly to complete this work. I apologize to them for not being able to mention them by name here; they are in my heart.

Maghrib Abidalreda Mahy Alramahi

Dedication

To the savior of mankind ...

Al-Imam Al-Mehdi (peace be upon him),

Who will fill the earth with justice and

equity after it has been filled with injustice and

oppression. The survival of religion in the earth.

To my father ...

My soul, who left me alone several years ago.

May God have mercy on you and make your abode in Paradise.

To my mother ...

You have always been a source of kindness, compassion, and constant

prayer.

To my wife and children ...

Who are the secret of my happiness in life.

To my brothers and sisters ....

Who supports me in this life.

Abstract

One significant challenge or problem in traditional networks is the method employed for load balancing. Typically, traditional networks rely on dedicated servers to perform load balancing tasks. These servers handle the complex task of distributing network traffic across multiple resources, and this approach has limitations in terms of flexibility and agility. The dissertation proposed using alternative modern approaches, such as Software-Defined Networking (SDN), to deal with the limitations present in traditional networks. Separating network control and data forwarding tasks establishes a centralized and programmed system to distribute network traffic based on dynamic conditions and policies intelligently.

The dissertation proposed a new approach to load balancing in SDN networks by proposing an integrated three-model algorithm, KNN-MLQLRL, which combines Machine Learning with Multilevel Queue and Load Balancing Scheduling (ML-MLQLBS). This integration aims to create an intelligent dynamic load balancing model, thus enhancing the network's Quality of Service (QoS) and workload distribution.

The proposed model comprises seven different stages. The initial phase encompasses generating, capturing, analyzing, and collecting packets to create a dataset, and the preprocessing stage is the second stage. Then, clustering methods are employed in the third stage to assign class labels. The fourth stage employs, the K-Nearest Neighbors (KNN) algorithm, to classify and predict priorities effectively. The fifth stage, on the other hand, uses the suggested Multilevel Queuing (MLQ) method to get priorities from the MLC and store them before sending them to servers. The proposed Load Balancing Scheduling (LBS) method is implemented in the sixth stage. This method uses a new algorithm called least Resource load (LRL) to determine which server has the least CPU and memory. Finally, the seventh stage is designing the proposed model. This model includes the results from the previous stages and intelligent dynamic load balancing to distribute work among servers better.

The proposed model was compared with six classical load-balancing algorithms based on four quality of service parameters: response time, latency, throughput, and load-balancing degree. The proposed model (KNN-MLQLRL) significantly improved in all parameters. The response time parameter in Dataset 1 achieved 20%, and in Dataset 2, it reached 15%. The latency parameter was 93% in Dataset 1, and in Dataset 2, it was 77%. The throughput parameter in Dataset 1 achieved 18%, and in Dataset 2, performance was 7%. Finally, the degree of load balancing parameter in Dataset 1 was 45%, and in Dataset 2, it was 38%.

Table of Contents

Section No.	Subject	Page No.
	Chapter One : General Introduction	
1.1	Overview	1
1.2	Problem Statement	3
1.3	Research Questions	4
1.4	Dissertation Aim and Objective	4
1.5	Dissertation Contribution	5
1.6	Related Work	6
1.6.1	Traffic Classification in SDN	6
1.6.2	Load Balancing Techniques in SDN	8
1.6.3	Load Balancing Metrics (LB- Metrics)	12
1.6.3.1	Analysis of the LB-Metrics	15
1.7	Dissertation Outline	17
	Chapter Two : Theoretical Background	
2.1	Overview	18
2.2	SDN Overview	18
2.2.1	The Design of SDN	19
2.2.2	OpenFlow Technology	21
2.2.3	The OpenFlow Architecture	21
2.3	Network Data Collection and Analysis Framework	23
2.3.1	Traffic Generation	23
2.3.2	Traffic Capturing	23
2.3.3	Traffic Analysis and Data Collection	24
2.4	Data Preprocessing	25
2.4.1	Data Cleaning	25

Section No.	Subject	Page No.
2.4.2	Data Integration	**25**
2.4.3	Data Transformation	**25**
2.4.4	Data Reduction	**26**
2.5	Applications of Machine Learning in SDN	**26**
2.5.1	Unsupervised learning (UL)	**27**
2.5.1.1	K-means Algorithm	**27**
2.5.1.2	DBSCAN Algorithm	**28**
2.5.1.3	OPTICS Algorithm	**28**
2.5.1.4	GMM Algorithm	**29**
2.5.2	Supervised learning (SL)	**30**
2.5.2.1	Decision Tree (DT) Algorithm	**31**
2.5.2.2	Naïve Bayes (NB) Algorithm	**32**
2.5.2.3	Support Vector Machine (SVM) Algorithm	**32**
2.5.2.4	K-Nearest Neighbors (KNN) Algorithm	**33**
2.5.2.5	Random Forest (RF) Algorithm	**33**
2.6	Evaluation Measures	**33**
2.6.1	Validation Set	**34**
2.6.2	Performance Metrics	**34**
2.6.3	Silhouette Score	**35**
2.7	Queue Scheduling	**36**
2.8	Parameters Calculation	**38**
2.8.1	Response Time (RT)	**38**
2.8.2	Latency (L)	**39**
2.8.3	Throughput	**39**
2.8.4	Degree of Load Balancing	**40**

Section No.	Subject	Page No.
2.9	The Research Dataset	41
Chapter Three : The Proposed Model		
3.1	Overview	42
3.2	Diagram of Proposed Model	42
3.3	Framework of Proposed Model	45
3.3.1	Stage 1 - The Data Generation	47
3.3.1.1	Request Generation	47
3.3.1.2	Traffic Capturing	47
3.3.1.3	Data Collection	47
3.3.1.4	Dataset Creation	47
3.3.2	Stage 2 - The Preprocessing	49
3.3.2.1	Data Reduction	49
3.3.2.2	Data Transformation	50
3.3.3	Stage 3 - Unsupervised Machine Learning (UML)	50
3.3.3.1	Proposed (OPTICS + GMM) Algorithm	51
3.3.4	Stage 4 - Supervised Machine learning (SML)	52
3.3.5	Stage 5 - Proposed Queue Scheduling	53
3.3.5.1	Proposed Model (ML-MLQ)	56
3.3.6	Stage 6 - Load Balancing Scheduling (LBS)	59
3.3.6.2	Proposed of Least Resource Load (LRL) Algorithm	63
3.3.6.2	Proposed Hybrid Algorithm: Least Connection and Least Resource Load (LRLLC) Algorithm	63
3.3.7	Stage 7 - Proposed Model (ML-MLQLRL)	66
Chapter Four : Results and Discussion		
4.1	Overview	70
4.2	System Requirement and Network Topology	70

Section No.	Subject	Page No.
4.3	Description of the SDN Network Datasets	**71**
4.4	The Results of the Preprocessing Stage	**73**
4.5	Evaluating the Clustering Algorithms	**75**
4.5.1	The Results of Datasets 1 and 2	**76**
4.6	Evaluating the Classification Model	**81**
4.6.1	The Results of Datasets 1 and 2	**82**
4.6.2	The Results of Performance Metrics	**84**
4.7	Evaluating the Queue Model	**88**
4.7.1	The Results of Multilevel Queue (MLQ) Model	**88**
4.7.2	The Results of KNN Model with Queue Model	**91**
4.8	Evaluating the Load Balancing Scheduling (LBS)	**93**
4.8.1	Proposal of Emulator and Network Topology	**94**
4.8.2	Evaluation Metrics in LB Scheduling Algorithms	**96**
4.8.3	The Results of Load Balancing Scheduling	**97**
4.9	Evaluating the Proposed Model	**104**
4.9.1	The Results of Proposed Model	**104**
Chapter Five : Conclusions and Future Works		
5.1	Conclusions	**111**
5.2	Future Works	**113**
References		**114**

List of Algorithms

Algorithm No.	Title	Page No.
3.1	Proposed (OPTICS + GMM) Algorithm	52
3.2	Proposed Multilevel Queue (MLQ) Model	55
3.3	Proposed K-NN with Multilevel Queue (MLQ) Model	58
3.4	Proposed of Least Resource Load (LRL) Algorithm	61
3.5	Proposed Hybrid Algorithm: Least Resource Load and Least Connection (LCLRL) Algorithm	64
3.6	Proposed Model (KNN-MLQLRL)	67

List of Figures

Figures No.	Title	Page No.
1.1	SDN vs Conventional Networks Explained.	2
1.2	Load Balancing Models: A/Traditional Network B/ SDN Network	3
1.3	Taxonomy of LB-Metrics	13
1.4	Percentage of LB-Metrics based on Table 1.5	16
1,5	Percentage of the simulation platforms used in SDN Network based on Table 1.4	16
2.1	Software-Defined Networking (SDN) Architecture	20
2.2	Open Flow Architecture	22
3.1	Proposed Model Diagram	43
3.2	Proposed Model Structure	44
3.3	Framework of Model Diagram	46
4.1	Create Datasets 1 and 2 Based on Requests (Types 1 and 2)	73
4.2	Feature Selection Scores Across Datasets	74

Figures No.	Title	Page No.
4.3	Dataset Standard Deviation Analysis Using Three Scaling Strategies	75
4.4	Visualization of Data Point Distribution in Datasets 1 and 2	78
4.5	Data Point Distribution: Clustering Algorithms - Datasets 1 & 2	81
4.6	Accuracy Comparison of Classification Algorithms in Datasets 1 and 2	84
4.7	Confusion Matrix Comparison of Classification Algorithms in Datasets 1 & 2	86
4.8	Evaluation Metrics Comparison of Classification Algorithms in Datasets 1 & 2	87
4.9	Pushing and Popping: Handling 10 Requests in MLQ	90
4.10	Exploring Push and Pop Operations with 20 Requests in MLQ	90
4.11	In-Depth Analysis: Managing 100 Requests through Push and Pop in MLQ	90
4.12	Testing KNN-MLQ Model with Push Operation	92
4.13	Testing KNN-MLQ Model with Pop Operation	92
4.14	Pushing and Popping: Efficient Handling of 29 Requests in the KNN-MLQ	93
4.15	Proposed Network Topology	96
4.16	Statistics of Response Time Parameter for Algorithms in Datasets 1 and 2	99
4.17	Statistics of Latency Parameter for Algorithms in Datasets 1 and 2	100
4.18	Statistics of Throughput Parameter for Algorithms in Datasets 1 and 2	102
4.19	Statistics of Degree of LB Parameter for Algorithms in Datasets 1 and 2	103
4.20	Statistics of Response Time Parameter for Algorithms in Dataset 1 and 2	106
4.21	Statistics of Latency Parameter for Algorithms in Dataset 1 and 2	107
4.22	Statistics of Throughput Parameter for Algorithms in Dataset 1 and 2	109
4.23	Statistics of Degree of LB Parameter for Algorithms in Dataset 1 and 2	110

List of Tables

Tables No.	Title	Page No.
1.1	Summary of the related works on traffic classification	8
1.2	Summary of the related works on load balancing techniques	11
1.3	Essential Research in LB- Metrics	13
1.4	Current Studies for LB-Metrics	14
1.5	Statistical analysis of LB-Metrics based on Table 1.3	15
2.1	Summary of Related Work for Classification Algorithms	31
2.2	Confusion Matrix	34
2.3	Types of Data and Files	41
3.1	Dataset Features	48
3.2	Datasets Dimension after using Feature Selection	49
4.1	Overview of the Model Requirement and Network Structure	71
4.2	Summary of File and Dataset Generation Details	72
4.3	Summary of Average Standard Deviation Results Using Three Strategies	75
4.4	Silhouette Score for Different Technique and Algorithms in Dataset 1	77
4.5	Cluster Distribution by Priority: Five Methods (Dataset 1)	80
4.6	Cluster Distribution by Priority: Five Methods (Dataset 2)	80
4.7	Classification Algorithm Evaluation for Dataset 1	83
4.8	Classification Algorithm Evaluation for Dataset 2	83
4.9	Classification Algorithm Performance Metrics for Dataset 1	85
4.10	Classification Algorithm Performance Metrics for Dataset 2	85
4.11	Percentage of the Essential Software Ecosystem used in the SDN Network	94

Tables No.	Title	Page No.
4.12	Overviews the Computer Components	95
4.13	Overviews the Topology Components and Emulator specifications	95
4.14	Statistics of Response Time Parameter for Algorithms - Datasets 1 and 2	98
4.15	Statistics of Latency Parameter for Algorithms - Datasets 1 and 2	99
4.16	Statistics of Throughput Parameter for Algorithms - Datasets 1 and 2	101
4.17	Statistics of Degree of LB Parameter for Algorithms - Datasets 1 and 2	102
4.18	Statistics of Response Time Parameter for Algorithms - Datasets 1 and 2	105
4.19	Statistics of Latency Parameter for Algorithms - Datasets 1 and 2	106
4.20	Statistics of Throughput Parameter for Algorithms - Datasets 1 and 2	108
4.21	Statistics of Degree of LB Parameter for Algorithms - Datasets 1 and 2	109

List of Abbreviations

Abbreviation	Meaning
API	Application Programming Interface
CL	Control Load
CM	Confusion Matrix
CPU	Control Process Unit
CV	Coefficient of Variation
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DC	Data Center
DCN	Data Center Networks
D-ITG	Distributed Internet Traffic Generator
DL	Deep Learning
DPI	Deep Packet Inspection
DT	Decision Tree
FN	False Negative
FP	False Positive
FTP	File Transfer Protocol
GMM	Gaussian Mixture Model
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
ID	Identifier
IP	Internet Protocol
K-NN	K-Nearest Neighbors

Abbreviation	Meaning
L	Latency
LB	Load Balancing
LBS	Load Balancing Scheduling
LC	Least Connection
LDA	Linear Discriminant Analysis
LR	Logistics Regression
LRL	Least Resource Load
ML	Machine Learning
ML-MLQ	Machine Learning of Multilevel Queue
MLQ	Multi-Level Queue
NB	Naive Bayes
NBG	Naive Bayes Gaussian
NL	Network latency
NN	Neural Network
OF	OpenFlow
OFS	OpenFlow Switches
OPTICS	Ordering Points To Identify the Clustering Structure
OSPF	Open Shortest Path First
PQ	Priority Queue
QoS	Quality of Service
R	Random
RAN	Radio Access Networks
RF	Random Forest

Abbreviation	Meaning
RL	Reinforcement Learning
RR	Round Robin
RT	Response Time
SDN	Software-Defined Networking
SL	Supervised Learning
ST	Service Time
SVM	Support Vector Machine
TCP	Transmission Control Protocol
Te-acc	Testing Accuracy
TN	True Negative
TP	True Positive
Tr-acc	Training Accuracy
UDP	User Datagram Protocol
UL	Unsupervised Learning
URL	Uniform Resource Locator
Wi-Fi	Wireless Fidelity
WRR	Weight Round Robin

List of Dissertation Related Publications

No.	**1**
Statues	Published / 2022
Name of Publisher	International Conference on Data Science and Intelligent Computing 2022 - (ICDSIC2022)
Sponsor	University of Kerbala - College of Computer Science and Information Technology, Iraq
Paper Title	Performance Analysis for Load Balancing Algorithms Using POX Controller in SDN
Details	IEEE, Scopus
Authors	Maghrib Abidalreda Maky Alrammahi Wesam S. Bhaya College of Information Technology, University of Babylon, Iraq
No.	**2**
Statues	Accepted for publication/ 2022
Name of Publisher	The 2nd International Conference on advances in Engineering Science and Technology 2022 - (AEST-2022)
Sponsor	University of Babylon, College of Engineering Babylon, Iraq
Paper Title	A State-of-the-Art Survey and Taxonomy for Load Balancing Metrics in SDN Networks
Details	IEEE, Scopus
Authors	Maghrib Abidalreda Maky Alrammahi Wesam S. Bhaya College of Information Technology, University of Babylon, Iraq
No.	**3**
Statues	Published / 2023
Name of Publisher	Journal of Al-Qadisiyah for Computer Science and Mathematics Vol. 15(3) 2023 , pp Comp. 164–182
Sponsor	University of Al-Qadisiyah, College of Computer Science , Iraq
Paper Title	A State-of-the-Art Survey and Taxonomy for Load Balancing Metrics in SDN Networks
Details	Locally
Authors	Maghrib Abidalreda Maky Alrammahi Wesam S. Bhaya College of Information Technology, University of Babylon, Iraq

Chapter One

General Introduction

1.1. Overview

The impact of technological progress on internet traffic has been significant. As such, more efforts are needed to enhance the network's intelligence, efficiency, and reliability [1]. The existing limitations in adapting to fast expansion in conventional networks require developing more resilient infrastructure to effectively accommodate such growth [2]. Network management challenges are frequently encountered in conventional networks due to the inherent vendor-specific characteristics of network hardware components, such as switches, routers, and load balancers. Therefore, these devices exhibit interconnected data and control planes. The adaptability of their functionality is constrained unless they use network management systems specific to the vendor [3].

To deal with the constraints associated with conventional networks, researchers are using alternative approaches like Software-Defined Networking (SDN) [4]. SDN can encapsulate network design and infrastructure within programmable software, enabling deployment across various hardware and devices [5]. The abovementioned software effectively addresses the limitations inherent in conventional network systems. Moreover, the concept of SDN brings about the capability of programmability within the network infrastructure, enabling the configuration and operation of the network to be defined and altered through programming techniques [6].

Figure 1.1 presents a comprehensive depiction of both traditional and software-defined network topologies within a unified framework. SDN enables decoupling network functions from hardware devices specific to a particular vendor. Separating the control plane, which manages the actual forwarding of data packets, from the data plane, which decides on network traffic [7].

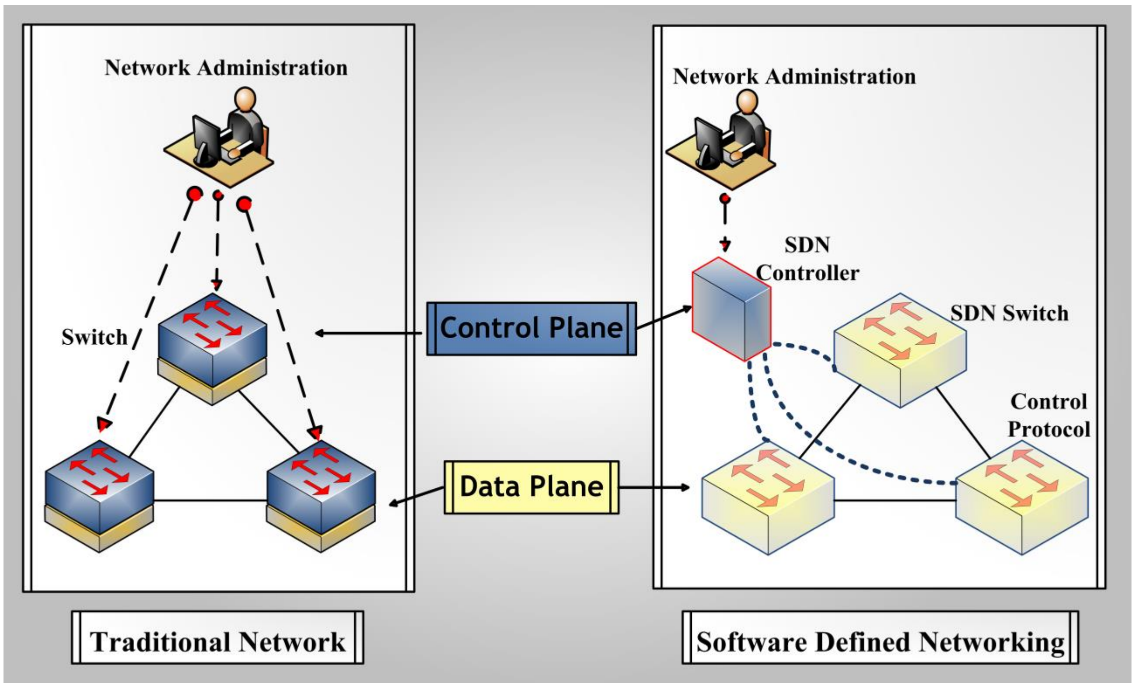


Figure 1.1 SDN vs Conventional Networks Explained [7].

SDN is a networking approach that separates the functionality of physical network devices and centralizes the decision-making process within the control plane, as illustrated in Figure 1. The control plane encompasses all aspects of network intelligence, including packet forwarding and establishing network administration policies [8]. The architectural design of SDN enables efficient network management and scalability [9]. Implementing an SDN controller facilitates the centralized management of the entire network, leading to the streamlining of network architecture and the separation of network control from vendor-specific dependencies [10].

Load balancing, the process of distributing traffic across network devices, holds significant importance in network design and management [11]. A proficient load balancer enhances network parameters, including latency, response time, resource utilization, and throughput. Traditional networks often employ a dedicated server to perform load balancing [12], as depicted in Figure 1.2. SDN load balancing can be set up in the SDN controller because it can be programmed. It makes it easier and more accurate to add or remove rules or policies from the SDN flow table [13].

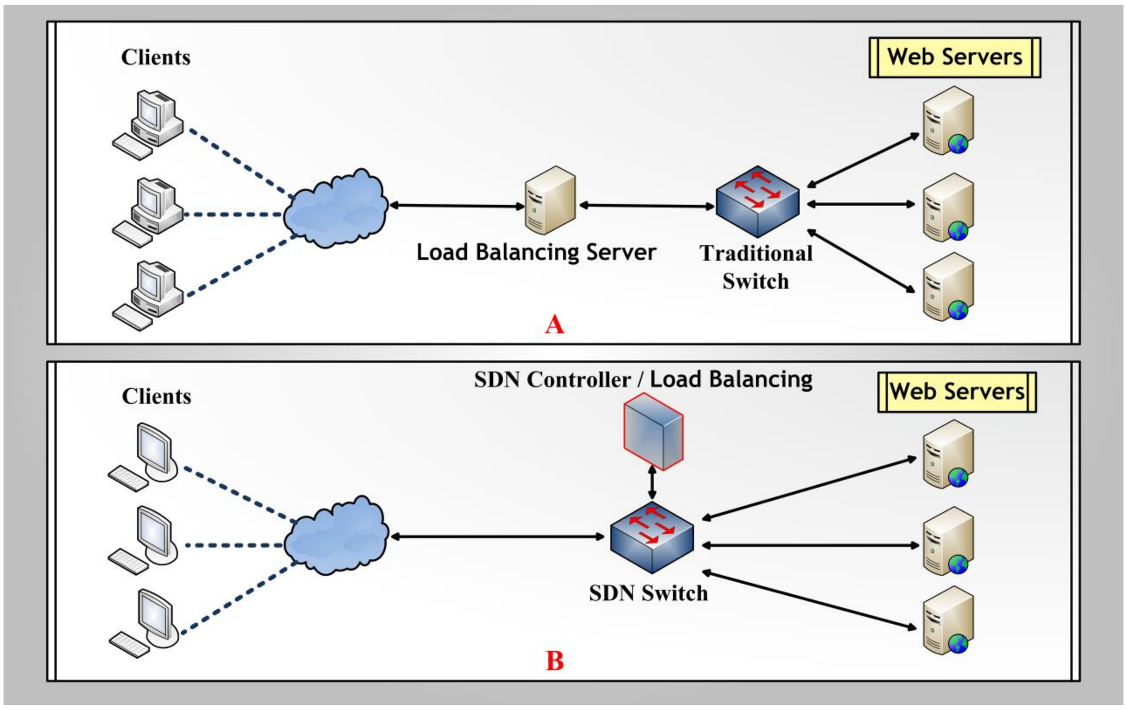


Figure 1.2 Load Balancing Models: **A**/Traditional Network **B**/ SDN Network [13]

1.2. Problem Statement

The primary challenge in SDN networks revolves around optimizing load balancing between servers and Quality-of-Service (QoS). It entails reducing response time and latency, increasing throughput, achieving a high degree of load balancing, enhancing resource location, and managing workload, among other servers.

1.3 Research Questions

This dissertation's primary question is how to build an intelligent model to improve dynamic load balance and QoS in SDN networks. From this question, many sub-questions can be extracted:

- What are the best techniques for clustering algorithms ?
- Why do use the queue model in SDN networks ?
- How can dynamic load balancing improve the quality of service ?
- How to build an intelligent model to improve dynamic load balance and QoS in SDN networks

1.4 Dissertation Aim and Objective

This dissertation aims to develop an intelligent dynamic load balancing model that can enhance the quality of service and distribute workloads efficiently among servers in SDN networks.

The objectives are to propose a model that reduces response time and latency, increase throughput, and optimize overall network performance. The achievement is attained by utilizing various objectives:

• Create and determine the suitable number of clusters, and discover and treatment of noise within a given dataset. This objective is achieved by the proposed of integration of the Optics and GMM algorithms.

• Addressing the problem of starvation of priorities. This objective is achieved by the proposed multilevel queue (MLQ) model.

• Select optimal servers, This is achieved by proposing a dynamic load-balancing scheduling algorithm based on the Least Resource Load (LRL) approach.

• Optimizing workload between servers and Quality QoS. This is achieved by Implementing an intelligent Dynamic load balancing model.

1.5 Dissertation Contribution

The analysis comprises seven different stages. The initial phase encompasses generating, capturing, analyzing, and collecting packets to create a dataset. Subsequently, the preprocessing stage is the second stage. Then, clustering methods are employed in the third stage to assign class labels. The fourth stage employs Machine Learning Classification (MLC), while the fifth stage utilizes the proposed Multilevel Queueing (MLQ) approach. The sixth stage implements the proposed Load Balancing Scheduling (LBS) technique. Lastly, the seventh stage involves the design of the proposed model, which incorporates the previous stages' findings and intelligent dynamic load balancing to enhance workload distribution across servers.

1. The proposed method integrates Optics and GMM algorithms. The first algorithm (Optics) creates clusters and detects noise in datasets. The second algorithm, GMM, processes noise. Integrating these two methods works to create datasets, including class labels.

2. It proposed a methodological contribution by implementing a multilevel queue (MLQ) model for organizing incoming network traffic. This system enhances queue management and minimizes delays by assigning different queues according to priority levels.

3. Additionally, it develops and proposes two algorithms based on the principles of dynamic load-balancing scheduling. The initial algorithm uses the Least Resource Load (LRL) strategy. The present model assesses network servers by considering their CPU and memory loads, with a selection for servers displaying the lowest load. The second algorithm integrates the Least Connection (LC) algorithm with the LRL approach to create (LCLRL).

4. The primary objectives of this dissertation are to enhance the performance of the SDN load balancing environment by reducing response time and latency and increasing throughput. Implementing the suggested intelligent model, which combines various algorithms with machine learning and a framework for multilevel queue and load balancing scheduling (ML-MLQLBS).

1.6 Related Work

This section explains the most important previous studies on traffic classification and load balancing techniques in SDN, as well as the most critical parameters and simulations used in SDN networks., and divide into three partitions:

1.6.1 Traffic Classification in SDN

This section reviews existing literature and examines the various methodologies and approaches employed in traffic classification. These methodologies involve data generation, analysis, collection, and pre-processing. In addition, it discovers and identifies the essential algorithms to create clustering for datasets, focusing on identifying the most crucial classification algorithms.

1- In this study, the authors [14] propose an SDN controller model with machine learning methods to detect and forward large traffic. This study will simulate elephant flow detection using supervised machine learning methods such as Naive Bayes (NB), K-Nearest Neighbors (KNN), Logistics Regression (LR), Support Vector Machine (SVM), and Decision Tree (DT). DT and KNN algorithms are the best machine learning methods for elephant flow detection, with 99% accuracy.

2- The authors [15] use feature extraction and classification techniques to analyze real-time traffic in SDN. Use of DDoS attack traffic to extract the 7 most relevant features and employed for training and testing classifiers. Several well-known classifiers, such as SVM, Random RF, KNN, XGBoost, and Naive Bayes NB. Specifically, the SVM achieves performance rates of 99.398%, 99.413%, 99.397%, 0.718%, 0.995, and 99.400% for these metrics.

3- In their publication, the authors [16] provide a traffic classification approach inside the architecture of a SDN. Six statistical features are included in the training of the Variational Autoencoder (VAE) through the use of Wireshark and Internet services. The suggested strategy has an accuracy rate of 89% on average. Comparing the remaining algorithms, it was discovered that they outperformed traditional statistics-based classification techniques like MLP, AE+MLP, VAE+MLP, and SVM by 52%, 47%, 39%, 59%, and 26%, respectively.

4- In this study, the authors [17] Create and implement a machine learning-based system that identifies mice and elephants early on. It made it possible to use Wireshark to record, analyze, and categorize network traffic before preprocessing and organizing the gathered information. The decision tree classifier proved to be the most successful algorithm, displaying a 100% confidence level, after the following machine learning models were trained: logistic regression, SVM, RF, linear discriminant analysis (LDA), KNN, Naive Bayes Gaussian (NBG), and DT.

5- The authors [18] the TCP/IP Traffic Flows dataset, which was used in this investigation, was downloaded from Kaggle. It uses a subset of six features that were taken out of the dataset to create a priority classification. It has 3,577,296 rows and 84 features in total. It made use of the RF, Extra Tree, KNN, and XGBOOST classification algorithms. The comparison of these methods showed accuracies of 97.5, 95.1, 93.6, and 79.5, in that order. Develop a machine learning-based proactive rerouting scheme (MLPRS) that uses dynamic load balancing in a real-time network topology to enhance the QoS. Compared to the default SDN configuration, the MLPRS has shown a significant improvement in performance, with a 3.7% increase in throughput and a 3.6% increase in bandwidth. Table 1.1 summarizes key findings from various studies on traffic classification in SDN environments.

Table 1.1. Summary of the related works on traffic classification

Ref.	Approach	Algorithms Compared	Dataset/ Environment	Result/ Accuracy	Pros	Cons
[14]	SDN-based traffic classification using machine learning	NB, KNN, LR, SVM, DT	Diverse network traffic from D-ITG	DT and KNN: 99%	Files are manually generated and not real	Generation of various applications in network traffic
[15]	Feature extraction and classification in real-time SDN traffic analysis	SVM, RF, KNN, XGBoost, NB	Traffic data with 7 features	SVM: 99.398%	Traffic representation based on seven features	Classification based on analysis of real-time traffic
[16]	Traffic classification in SDN using VAE	MLP, AE+MLP, VAE+MLP, SVM	Statistical features of network flows	Proposed method: 89%	Utilizes limited features	Statistical features collected of network flows from real-world
[17]	Machine learning-based system for early detection of elephants and mice in SDN	LR, SVM, RF, LDA, KNN, NBG, DT	Network traffic data	Decision tree classifier: 100%	Binary classification (mice or elephants)	System based on a calculation of dynamic threshold
[18]	ML-based proactive re-routing scheme for QoS improvement in SDN	RF, Extra Tree, KNN, XGBOOST	Kaggle, TCP/IP Traffic flows	RF: 97.5%	Calculate only two ,metrics (Throughput, Bandwidth)	Classification based on analysis of real-time traffic

1.6.2 Load Balancing Techniques in SDN

Section two reviews existing literature and examines the modern methodologies and approaches employed in load balancing for SDN. These methodologies include knowledge of the latest proposed methods and algorithms in handling load distribution in SDN networks.

1- In this study, the authors [19] introduced a new technique called the Multiple Regression-Based Searching (MRBS) algorithm with the goal of improving the Data Center Networks' (DCNs') server selection and routing paths' performance. This analytical technique utilizes server data parameters, including load, response time, bandwidth, and server usage. The algorithm that is being considered mitigates time and delay by over 45 %. It shows an 83% increase in server usage over traditional techniques.

2- The authors [20] provided a theoretical foundation for load-balancing, scaling, and monitoring modules used in SDN networks. To help OpenFlow switches (OFS) in the task of load-balancing network traffic to and from virtual network function (VNF) clusters, the suggested method makes use of the SDN controller. The CPU's load-balancing strategy successfully reduced imbalances, with an average variation of less than 10%.

3- In their publication, the authors [21] To optimize the routing strategy for SDN, introduce GNN-DRL, an intelligent routing algorithm. In comparison to OSPF, ECMP, and EARS intelligent routing algorithms, the comparative analysis results show that GNN-DRL achieves a reduction of 13.92% in maximum link utilization and a decrease of 9.48% in end-to-end delay.

4- In this study, the authors [22] recommend utilizing deep reinforcement learning (DRL) to enhance SDN network routing optimization by determining the optimal link weights for effective network traffic distribution, reducing delay and packet losses The authors propose an M/M/1/K queue network model as a solution to the problem of DRL's extended learning time due to topology changes. The improvement is apparent in lower end-to-end latency and reduced packet loss at switches, according to the simulation results.

5- In this study, the authors [23] explain the use of Improved Ant Colony Optimization (IACO) in SDN networks to manage dynamic load balancing. Flow table distribution, network topology awareness, status collection, and the basic load-balancing algorithm are the four separate modules that together make the structure of the load-balancing system, which is based on the IACO. By dynamically modifying the routing strategy in response to variations in network link traffic and server utilization, the results demonstrated that IACO-LB effectively manages the load balancing problem.

6- The authors [24] Provide a new method based on a heuristic that combines deep learning and reinforcement learning methods. Finding the perfect amount of controllers and locations dynamically is a part of it. The main objective is to reduce the Control Load (CL), Intra-Cluster Delay (ICD), and Intra-Cluster Throughput (ICT) as well as the controller's response time, which is defined as the time delay between the control panel and the switches. In particular, the experimental results improved response time and resource usage, which improved network performance.

7- In this study, the authors [25] propose a new technique for enhancing bandwidth management in peer-to-peer applications running on SDN networks by utilizing the Random Forest algorithm. When compared to the existing bandwidth allocation techniques, the experimental results indicate that the proposed framework has the capability to greatly improve the QoS. In terms of success rate, throughput, response time, etc.

8- The authors [26] provide a description of load balancing scheme called Load Balancing by Optimizing Resource Utilization (LBORU). This scheme uses multi-parameter metrics, such as CPU load, I/O Read, I/O Write, Link Upload, and Link Download, to schedule connections and monitor real-time server load indicators. Experiments indicate that the LBORU mechanism, which keeps track of CPU values and network resource demand, performs better when related to server load balancing than the other methods (R, RR, LBBSRT).

9- In this study, the authors [27] propose a new genetic load balancing algorithm (GLBA) that is specifically developed for use in multimedia applications. When compared to alternative algorithms (WRR, dynamic server, and LBBSRT), the suggested technique reduces end-user response time while increasing throughput by efficiently distributing the load across numerous servers. Table 1.2 summarizes key findings from various studies on load balancing techniques in SDN environments.

Table 1.2. Summary of the related works on load balancing techniques

Ref.	Approach	Algorithms Compared	Dataset/ Environment	Result/ Accuracy	Pros	Cons
[19]	Multiple Regression-Based Searching (MRBS) algorithm for server selection and routing path optimization in DCNs	Conventional algorithms	Data Center Networks (DCNs)	MRBS: Delay and time reduction by 45%, Server utilization 83%	Calculate only two metrics	Traffic prediction, server utilization
[20]	Theoretical framework for SDN with monitoring, scaling, and load-balancing modules	SDN controller, OpenFlow switches	Testing and experimentation	Load balancing with <10% average deviation	QoS parameters are not calculated	The system incorporates adaptive load balancing, scalability, and monitoring.
[21]	GNN-DRL intelligent routing algorithm for SDN	OSPF, ECMP, EARS	Network topology data	GNN-DRL: -13.92% link utilization, -9.48% end-to-end delay	Calculate only two metrics	Utilizing an intelligent alg orithm for route calculation
[22]	SDN and DRL-based routing optimization	conventional hop-count routing, a traffic demand-based RL algorithm	Simulation with different network topologies	Improved network performance	Calculate only two metrics	Enhance the end-to-end delay and minimize packet loss.
[23]	Improved Ant Colony Optimization (IACO-LB) for dynamic load balancing in SDN	Multi-path and path server traffic scheduling algorithms	Network connection and server metrics	IACO-LB successfully handles load balancing	load balancing based on the server side only	Dynamic load balancing system comprises four distinct modules

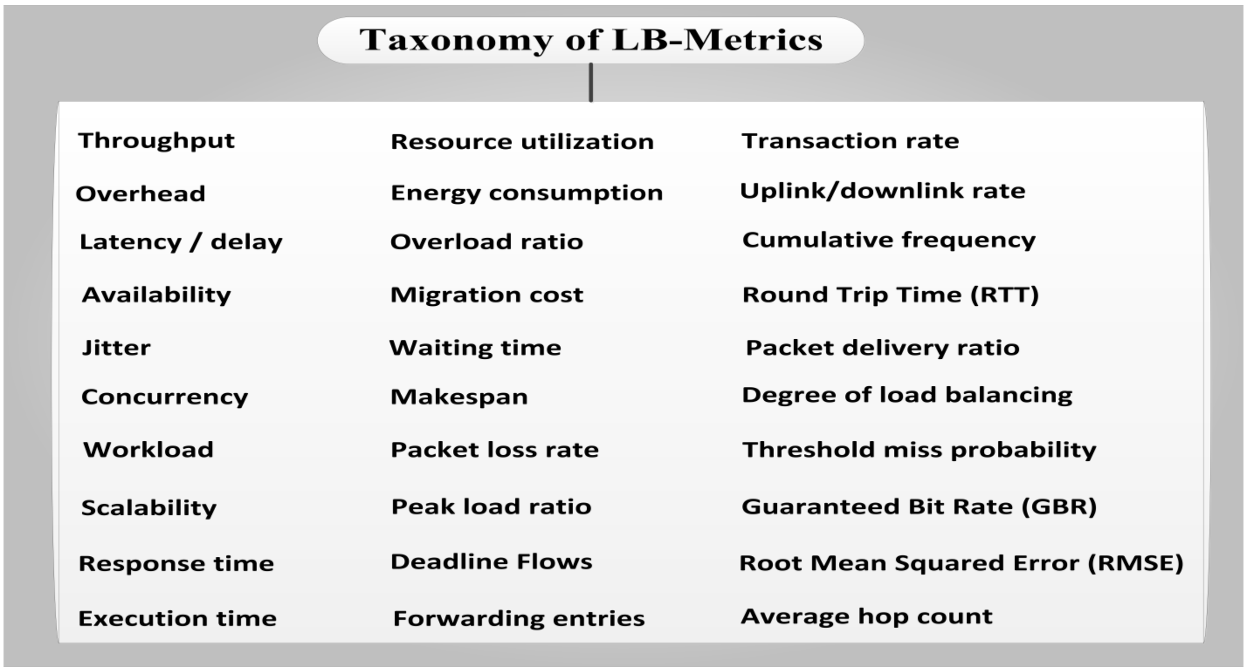
[24]	Heuristic with reinforcement learning and deep learning	N/A	Multi-Controller and Data Center	Proposed framework improves response time and resource utilization	Calculate only two metrics	Involve the utilization of multiple controllers.
[25]	Random Forest algorithm for bandwidth management in P2P applications using SDN	Bandwidth allocation algorithms	Bandwidth usage, network reconfiguration	Proposed framework improves QoS	Calculate only one metric	Improve technique allows for the decentralized distribution of data.
[26]	Load balancing scheme using real-time server load indicators and multi-parameter metrics	CPU load, I/O Read, I/O Write, Link Upload, and Link Download	Data Center include Multi Servers	Outperforms existing load-balancing schemes	load balancing based on the server side only	Workload distribution efficiency, employing multi-parameter metrics
[27]	Genetic load balancing algorithm (GLBA) for multimedia applications	Server load, weighted round robin, dynamic server, LBBSRT	Data Center include Multi Servers	GLBA aims to enhance throughput and reduce response time	load balancing based on the server side only	Dynamic load balancing

1.6.3 Load Balancing Metrics (LB- Metrics)

The subsequent section, will discuss the present evaluation of load balancing. The findings presented in Table 1.3 from extensive research on LB-Metrics indicate that the taxonomy for the parameter category includes 30 parameters, as shown in Figure 1.3.

Table 1.3 Essential Research in LB- Metrics

S	Ref	Related only about LB-Metrics					
		Taxonomy	Parameters Number	Summary	Current Studies	Percentage Analysis Chart	Simulations Platform
1	[28]	×	9	√	×	×	×
2	[29]	×	19	√	√	√	√
3	[30]	√	16	×	√	√	×
4	[31]	×	21	√	√	√	×
5	*proposed*	*√*	*30*	*√*	*√*	*√*	*√*


Taxonomy of LB-Metrics

Throughput	Resource utilization	Transaction rate
Overhead	Energy consumption	Uplink/downlink rate
Latency / delay	Overload ratio	Cumulative frequency
Availability	Migration cost	Round Trip Time (RTT)
Jitter	Waiting time	Packet delivery ratio
Concurrency	Makespan	Degree of load balancing
Workload	Packet loss rate	Threshold miss probability
Scalability	Peak load ratio	Guaranteed Bit Rate (GBR)
Response time	Deadline Flows	Root Mean Squared Error (RMSE)
Execution time	Forwarding entries	Average hop count

Figure 1.3: Taxonomy of LB-Metrics

The current study investigates an important part, including the current research on metrics employed in load balancing for SDN networks, which includes 41 research papers, as presented in Table 1.4

Table 1.4: Current Studies for LB-Metrics

Reference	Throughput	Overhead	Latency / delay	Availability	Jitter	Concurrency	Workload	Scalability	Response time	Execution time	Resource utilization	Energy consumption	Overload ratio	Migration cost	Waiting time	Makespan	Packet loss rate	Peak load ratio	Deadline Flows	Forwarding entries	Transaction rate	Uplink/downlink rate	Cumulative frequency	Round Trip Time (RTT)	Packet delivery ratio	Degree of load balancing	Threshold miss probability	Guaranteed Bit Rate	Root Mean Squared Error	Average hop count	Frameworks
[32]		*										*																			MATLAB
[33]			*	*																					*						Python
[34]	*	*					*			*																					Mininet/ Java
[35]	*										*																				Mininet /C++
[36]	*														*	*															General
[37]			*																							*	*				MATLAB
[38]		*	*																												Mininet/ Python
[39]									*												*										Mininet/ Python
[40]	*								*												*										Mininet/ Python
[41]	*			*		*			*								*				*										Mininet/ Python
[42]			*																			*									Mininet/ Python
[43]			*									*																			Juniper/C++
[44]									*	*																					General
[45]																						*									HetNet
[46]			*									*																			iFogSim/Java
[47]			*		*												*														Mininet/Java
[48]	*	*			*												*							*							Mininet/Java
[49]	*																				*										Mininet/Java
[50]									*		*										*										Mininet/Java
[51]									*		*					*															Cloudsim/Java
[52]			*					*																							Mininet/Java
[53]	*							*									*									*					Mininet/Java
[54]														*												*					Python
[55]									*		*																				Mininet/Java
[56]	*	*																								*					SimPy/ Python
[57]		*															*		*				*								Mininet/Java
[58]		*															*												*		Mininet/ Python
[59]																				*											Not Clear
[60]																							*								Use of Statistics
[61]													*													*		*			NS-3/ Java
[62]		*					*			*																					C++
[63]		*																*													Mininet/Java
[64]									*						*	*										*					Cloudsim/Java
[65]		*							*							*															Not Clear

[66]												*			*																MATLAB
[67]	*										*																				C
[68]		*																								*					Mininet/Java
[69]	*		*		*				*					*			*									*					Mininet/Java
[70]		*																						*		*					Mininet/Java
[71]							*																			*					Mininet/ Python
[72]	*		*																						*					*	NS2/Java

1.6.3.1 Analysis of the LB-Metrics

It is imperative to understand the key parameters that the dissertation must consider when evaluating network performance following load balancing implementation in SDN networks. Based on the data presented in Table 1.4, Table 1.5 provides statistical information for the parameters. In general, eight parameters of highest significance and extensive utilization have emerged.

Table 1.5: Statistical analysis of LB-Metrics based on Table 1.3

S	Metrics	Total of Use	Percentage
1	Throughput	12	10.6%
2	Overhead	12	10.6%
3	Degree of LB	11	9.7%
4	Latency	10	8.8%
5	Response Time	9	8%
6	Packet Loss Rate	7	6.2%
7	Resource Utilization	5	4.4%
8	Transaction Time	5	4.4%
9	Others	42	37%

In Figure 1.4, the current studies illustrate LB-Metrics distribution, including Throughput, Overhead, Degree of LB, Latency, Response Time, Packet Loss Rate, Resource Utilization, Transaction Time, and Others. These metrics are included in the ratios: 10.6%, 10.6%, 9.7%, 8.8%, 8.0%, 6.2%, 4.4%, 4.4%, and 37%, respectively. According to Table 1.4, Figure 2.8 depicts the simulation platforms and percentages for LB-SDN.

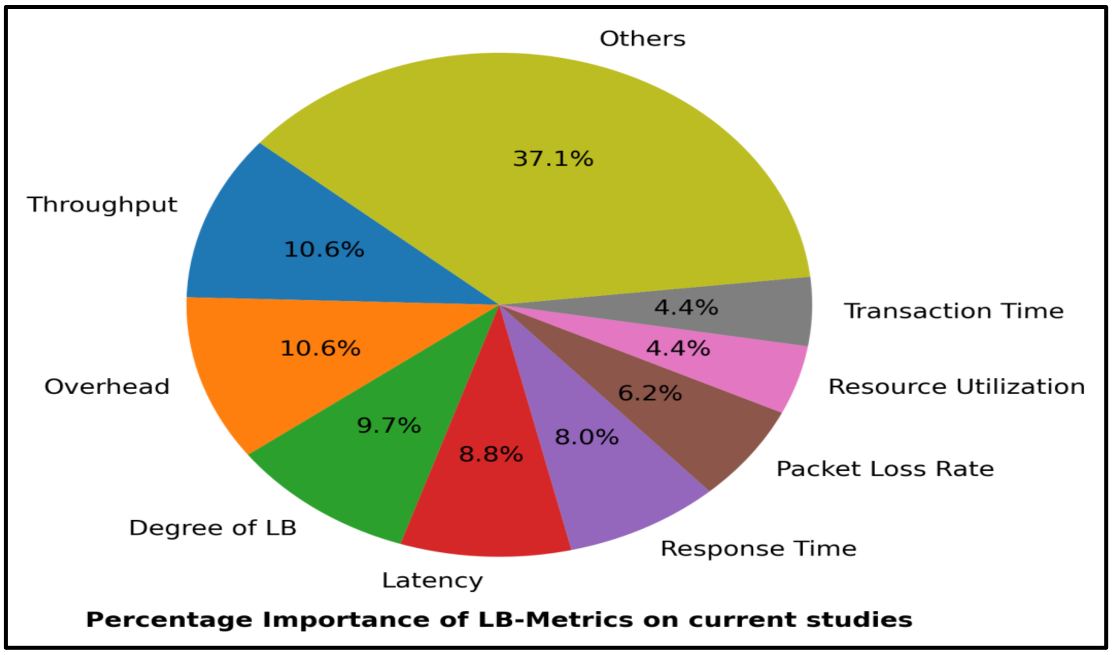


Figure 1.4:Percentage of LB-Metrics based on Table 1.5

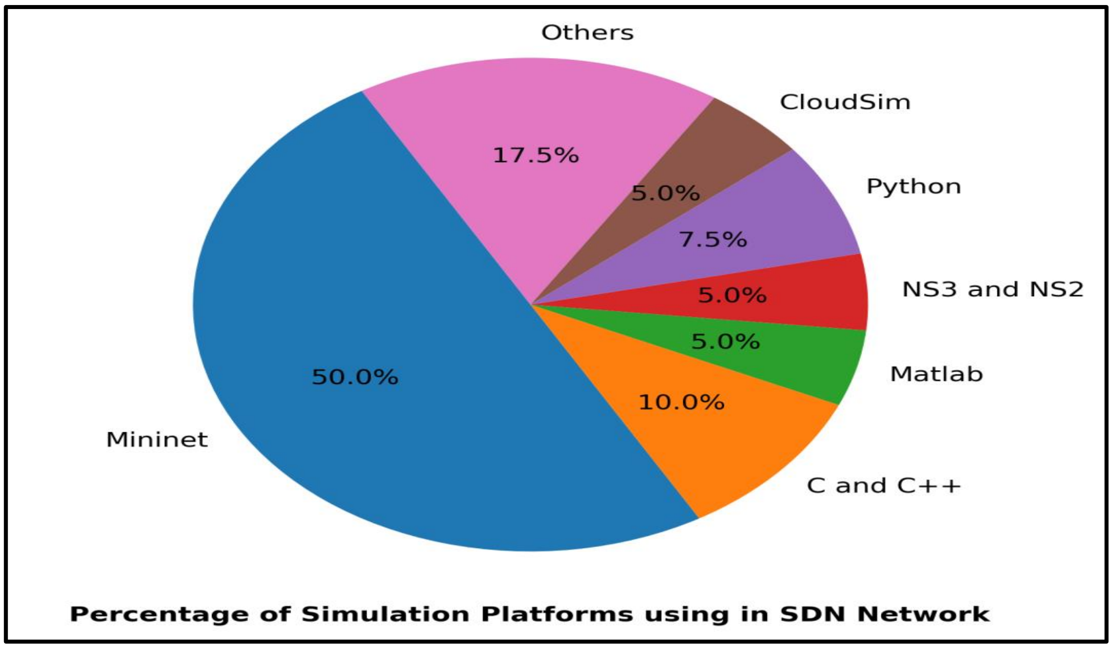


Figure 1.5:Percentage of the simulation platforms used in SDN Network based on Table 1.4

1.7 Dissertation Outline

The subsequent sections of the dissertation are organized as follows:

Chapter Two: Presents a comprehensive overview of traffic classification, preprocessing, and load Balancing (LB) principles. A comprehensive overview of each unit's critical clustering and classification algorithms, the primary factors influencing their effectiveness, and a selection of evaluation metrics employed.

Chapter Three: Comprehensively explains the primary procedures involved in data generation and collection, including preprocessing and creating dataset labels. It also discusses the development of a learning model based on machine learning algorithms, as well as the design of an integrated model that combines Machine Learning (ML) with Multilevel Queue modeling (MLQ) and Load Balancing Scheduling (LBS). This integrated model, ML-MLQLBS, is proposed as a solution.

Chapter Four: It analyses the utilization of the proposed model on two datasets. The first dataset is generated using unreal files, while the second uses real files from Facebook. This chapter also presents the discussion and experimental results of implementing the proposed model on these datasets.

Chapter Five: This dissertation presents the conclusions and recommendations for future work.

Chapter Two

Theoretical Background

2.1 Overview

This chapter introduces the theoretical concepts in this dissertation by presenting the concept of SDN and architecture design and discussing the necessary OpenFlow protocol. Explain stages of network data collection and data preprocessing. Highlighting the most critical clustering and classification algorithms, and an overview of the different methods used to assess the effectiveness of machine learning models such as validation set, calculating of accuracy, and performance metrics. Discussion of types examining traffic management in an SDN comprehensively. Analyzes recent research studies and the taxonomy of LB in SDN, such as classification, algorithms, techniques, and metrics. This research provides a comprehensive, state-of-the-art survey of LB in SDN according to LB-Classification, LB algorithms, LB-Techniques, and LB-Metrics. Finally, explain the mathematical methods used for the computation of significant parameters.

2.2 SDN Overview

SDN is a modern networking framework that reduces network management using a programmable software controller [73]. According to [74], the central view can enhance the controller's ability to efficiently and flexibly manage network flows.

SDN eliminates the need for separate control and data planes by implementing a single software controller capable of managing the entire network [75]. The controller can make decisions more directly and efficiently distribute these decisions to forwarding devices [76]. SDN protocols such as ForCES, OpenFlow, and HyperFlow facilitate the transmission of control packets from the controller to the switches, which allows the controller to control its logic [77].

2.2.1 The Design of SDN

The described network architecture enhances programmability and flexibility by separating network management, routing, and switching tasks from physical hardware components, such as routers and switches . SDN management is crucial in enhancing network configuration, performance, and monitoring capabilities [78], as depicted in Figure 2.1.

The SDN architecture is composed of the following components:

The application plane represents a network's applications, which serve as user interfaces and SDN applications [79].

The SDN architecture facilitates the rapid and effortless development of supplementary services and applications, thereby enabling the generation of new services [80]. This layer encompasses load balancing, routing, policy enforcement, and other specialized applications for enhancing network resource services.

The control plane includes the layer where SDN intelligence is housed. The SND controller is responsible for managing and controlling the various operations and functionalities of the network. For example, the controller is responsible for executing all policies, making suitable decisions regarding packet forwarding, and implementing infrastructure rules [80]. The control plane is typically characterized by its conceptual centralization, centralized operation, and physical dispersion, aiming to enhance scalability and reliability [81]. The Northbound application programming interfaces (APIs) facilitate communication between the application and control planes [82]. Application programming interfaces (APIs) frequently exhibit an open-source nature, enabling enhanced adaptability as they can be consistently changed to adapt to particular needs .

The data plane, or infrastructure plane, includes the network equipment that facilitates data transmission, including routers, switches, servers, and bridges [83]. Southbound application programming interfaces (APIs) make it easier to connect the most frequently used data and control planes of the OpenFlow protocol [84].

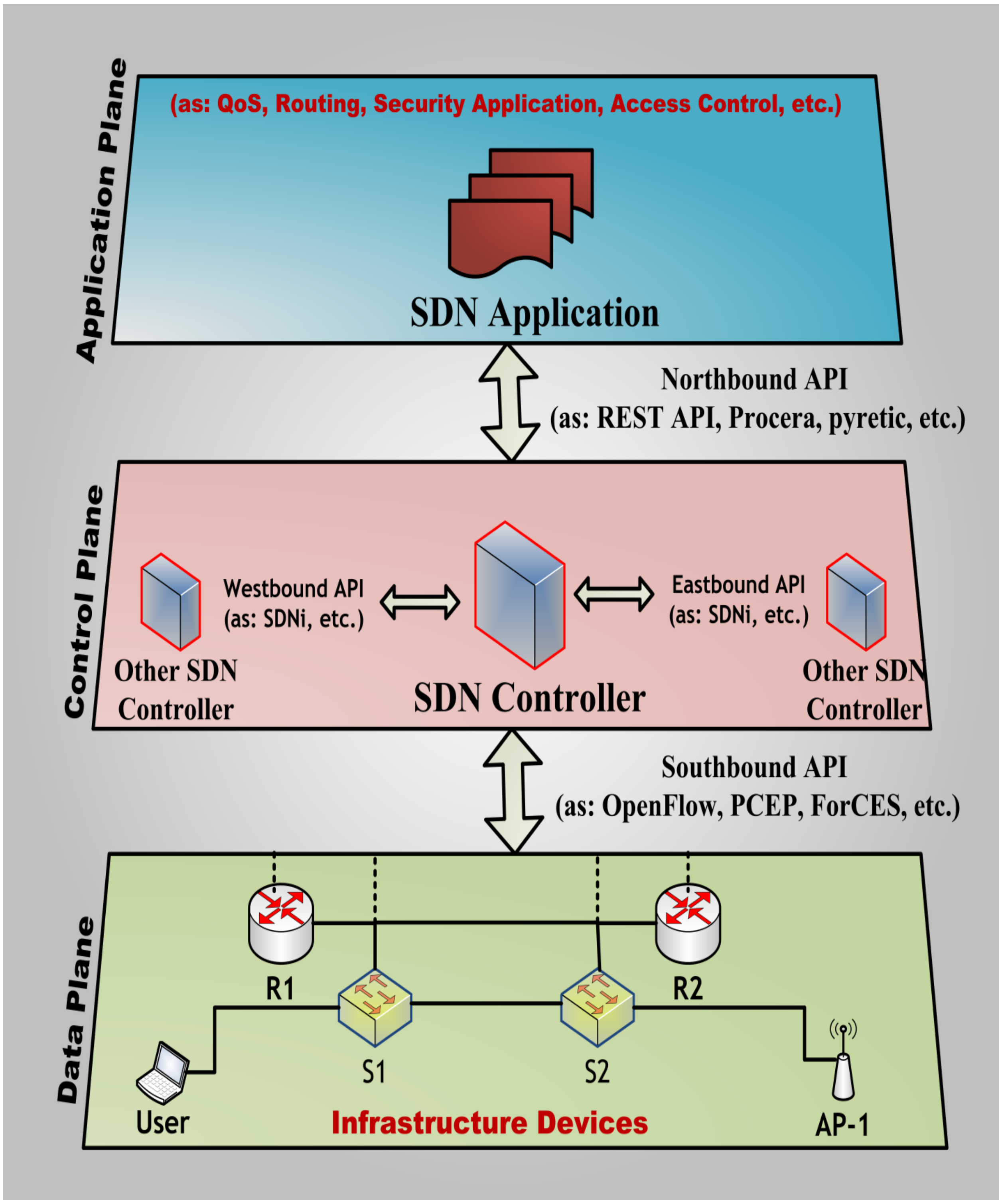


Figure 2.1: Software-Defined Networking (SDN) Architecture [84]

2.2.2 OpenFlow Technology

OpenFlow is a set of protocols that facilitate enhanced communication on both the control and data planes [85]. The standardization of OpenFlow was first carried out by the Open Networking Foundation (ONF), subsequently leading to its adoption by significant players in the information and communications technology (ICT) sector, such as Cisco, HP, and Google [86].

OpenFlow enables the SDN controller to manage the distribution of network traffic effectively, implement load balancing mechanisms, enforce security measures, and ensure Quality of Service (QoS) across data plane devices [87].

The southbound application programming interface (API) facilitates the exchange of information between devices that are connected to SDN controller [88]. OpenFlow enables the direct configuration of network devices and switches. For example, OpenFlow facilitates the transmission of network flow regulations to switches and enables access to the flow table [89]. The SDN controller facilitates rapid network modifications due to its programmable nature.

2.2.3 The OpenFlow Architecture

The OpenFlow protocol is widely recognized as one of the most essential protocols in SDN [89]. The OpenFlow network architecture consists of three fundamental components: an OpenFlow controller, an OpenFlow switch, and an OpenFlow protocol [90]. Figure 2.2 depicts how information is exchanged between a SDN plane and a data plane.

The OpenFlow protocol facilitates the exchange of information by separating network components . Upon receiving a packet, an OpenFlow switch proceeds to examine the contents of the packet's header and then compare them to corresponding fields within the entries of the flow table [91]. If the packet header corresponds to an entry, the switch will conform to the directed instructions or actions specified in the flow entry, which may include the forwarding of packets. If a corresponding entry is not found, the switch will execute the instructions specified in the flow entry for table miss [92].

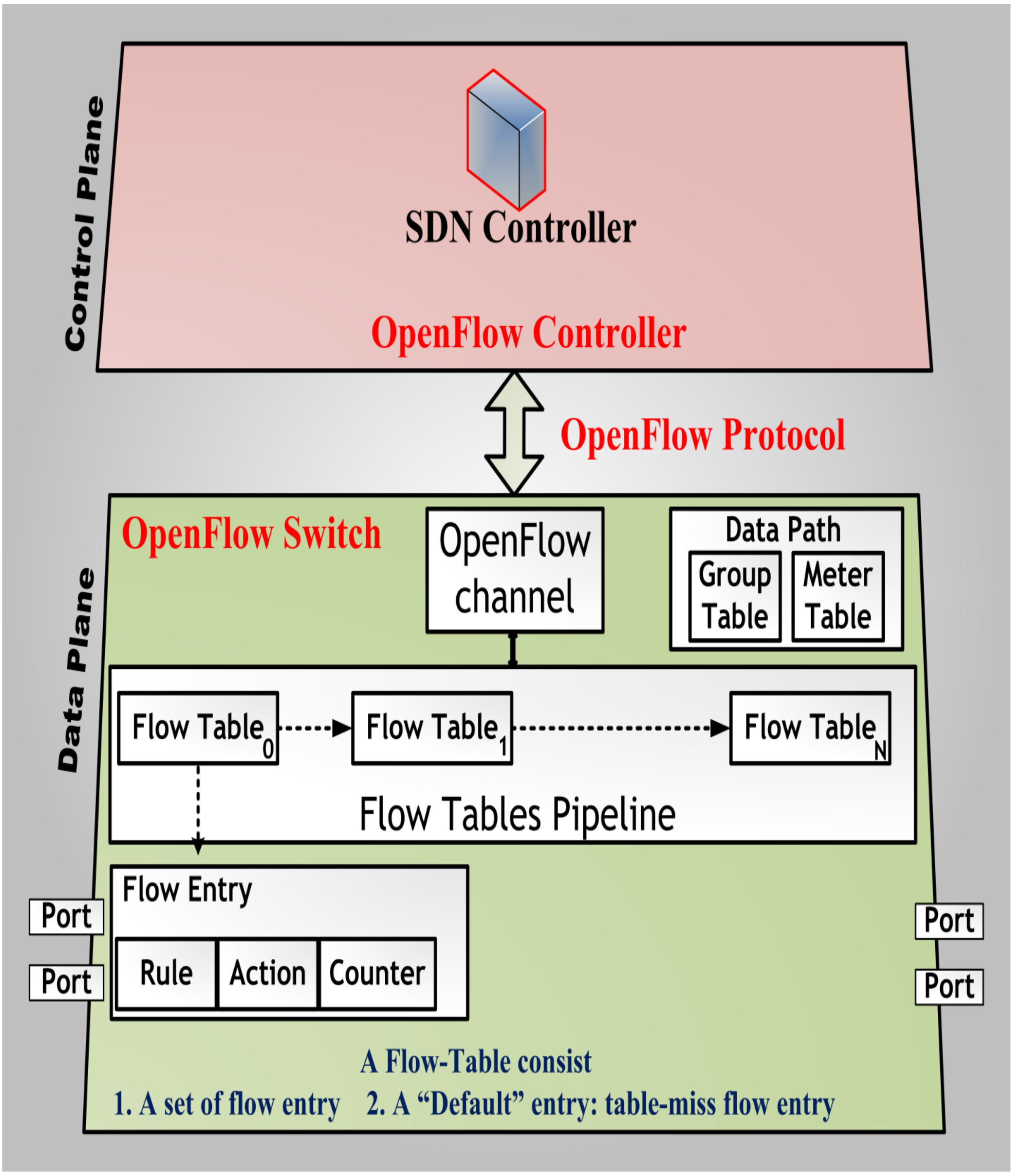


Figure 2.2 : Open Flow Architecture [90]

2.3 Network Data Collection and Analysis Framework

The utilization of various methodological frameworks enables researchers to construct significant datasets that effectively show network behavior in real-world scenarios. The method includes three fundamental stages: traffic generation, traffic capture, and traffic analysis and data collection.

2.3.1 Traffic Generation

Two distinct categories of file transfers exist, namely static files and dynamic files [93]. The initial category pertains to situations where the data size is predetermined and remains constant, such as a 1-megabyte or video file. The second category refers to instances where changes occur in the size or content, such as including a query parameter within a URL or the execution of a database query.

2.3.2 Traffic Capturing

The classification of internet traffic holds significant importance in network management, security, and quality of service (QoS) optimization. Over the years, various methodologies have been devised to efficiently categorize internet traffic [94]. The researcher provides an analysis of several prominent and extensively utilized methodologies for classifying internet traffic that are used in packet capture and classification into four types:

- **Port-Based Approach:** The port-based approach is widely regarded as the oldest and most prevalent method for traffic classification. This approach involves analyzing the communication ports utilized in the TCP/UDP header and establishing their correlation with established TCP/UDP port numbers [95].

- **Statistical Approach:** Statistical classification methods such as Wireshark are employed to analyze network traffic by leveraging various statistical features and metrics. Characteristics encompass packet size, inter-arrival times, packet counts, and flow duration [96]. Machine learning algorithms and statistical techniques are commonly employed in constructing models capable of classifying traffic based on the features above [97]. Statistical methods prove highly advantageous in analyzing extensive network data, necessitating automated classification techniques.

- Pattern Matching Approach: The classification process entails identifying distinct patterns or signatures within network traffic to determine the specific application or protocol. The methodology employed in this approach is based on pre-established patterns or rules that align with recognized applications or services. Deep packet inspection (DPI) is a commonly utilized technique in pattern matching, whereby the contents of packets are carefully examined to find distinct patterns exclusive to specific applications [98].

- Protocol Decoding Approach: The protocol decoding methodology entails the examination of the content and structure of network packets to determine the underlying protocols being utilized. The process frequently involves analyzing packet headers and payloads to identify distinct protocols. This approach can identify protocols, such as TCP, UDP, HTTP, FTP, etc., by analyzing the header information [99].

2.3.3 Traffic Analysis and Data Collection

Following the capture of the packets, they are subjected to analysis to extract valuable information and gain insights. In this phase, the collected data is analyzed, wherein diverse attributes are identified, including traffic patterns, utilized protocols, packet sizes, error rates, and other pertinent statistical measures. The analysis process aids in comprehending the behavior of networks, recognizing deviations from the standard, and identifying potential issues or weaknesses [100].

2.4 Data Preprocessing

Data preprocessing is a crucial step in data analysis, as it involves transforming raw data into well-structured datasets. This transformation is necessary to ensure that data analytics techniques can be effectively applied to the data. The dataset requires data preprocessing techniques to eliminate irrelevant and redundant data.

Data preprocessing is the initial and crucial stage in data analytics. The most crucial preprocessing stages involve data cleaning, integration, transformation, and reduction. The primary objectives of data preprocessing involve removing irrelevant data and establishing consistency in data representations [101].

2.4.1 Data Cleaning

It refers to discovering and fixing errors, inconsistencies, and errors within a given dataset. It could include duplicate events. Data cleaning techniques such as filling in missing values, identifying outliers, removing inconsistent data, and removing noise are essential in data analysis so that analysts can verify the accuracy and consistency of the information, thus improving the accuracy and value of their research [101].

2.4.2 Data Integration

It involves systematically combining data from various sources into a single file. Diverse sources frequently exhibit variations in data formats and designations. Data integration tackles these gaps to create a comprehensive dataset. This procedural measure helps prevent the occurrence of data gaps and facilitates a comprehensive examination of the information at hand [101].

2.4.3 Data Transformation

Refers to the systematic process of converting raw data into a structured format suitable for utilization in research tasks. This process includes various techniques such as normalization of numerical values, encoding categorical variables, and applying mathematical transformations to enhance the interpretability and usability of the data. Modifying the data ensures that all variables are subjected to equal and consistent treatment throughout the research [101].

2.4.4 Data Reduction

It aims to decrease the size of a dataset while retaining its essential information. The existence of a significant quantity of information that could create challenges for computational processing may produce beneficial results. Methods such as feature selection and dimensionality reduction aid in the simplification of information while retaining crucial insights [101].

2.5 Applications of Machine Learning in SDN

SDN has significantly transformed network flexibility, agility, and programmability. Machine learning (ML) is a methodology employed within SDN architecture to enhance network functionalities and non-functional parameters, including performance, security, etc. ML is a theoretical framework containing three fundamental elements: the model, parameters, and the learning system. The model is capable of making predictions and identifying patterns. The parameters refer to the signals or factors that the model utilizes to enhance its performance in terms of prediction or classification. The learning system is a computational framework for training, evaluating, and testing models using designated training and test sets [102].

Machine learning (ML) provides decision-making capabilities for computing systems, enhancing their intelligence. This capability can be utilized in SDN across various applications, particularly in the control layer, where it serves as the decision-making entity within the SDN architecture [103].

Machine learning (ML) has been employed to enhance network performance, improve security measures, optimize Quality of Service (QoS), and deal with various ineffective elements within the framework of SDN.

2.5.1 Unsupervised learning (UL)

One significant distinction between the Unsupervised learning (UL) and Supervised learning (SL) models is based on their utilization of known values as supervisory signals. SL is a machine-learning approach that utilizes labeled data to determine patterns and train a model to assign labels to data. This approach is frequently utilized when labeling costs are high or labeling could be more helpful. UL can be divided into three broad categories in data analysis: clustering, signal decomposition, and neural networks [104].

Clustering is a well-established subfield of unsupervised learning (UL) that aims to identify distinct subgroups within a raw, unlabeled dataset based on similarities and dissimilarities in their features. Various clustering techniques are classified based on different criteria. These include Partitioning Algorithms, such as K-means; Density-Based Algorithms, such as DBSCAN and OPTICS; Hierarchical Algorithms, such as BIRCH; and Distribution-Based Algorithms, such as Gaussian Mixture Models (GMM), among others [105].

2.5.1.1 K-means Algorithm

The K-means algorithm is widely recognized as a prominent iterative clustering method in partitioning algorithms. The K-means algorithm iteratively adjusts the cluster centers to minimize the total within-cluster variance based on the user's selection of the desired number of cluster centers [106]. The K-Means algorithm necessitates the specification of the number of clusters as an input parameter, and it exclusively identifies spherical clusters. This shape is not conducive to representing reality [107].

Limitation:

- Specifying the number of clusters ('k') is necessary.

- The sensitivity to the initial location of centroids can result in varying outcomes across multiple runs.

- The assumption made in this study is that the clusters are spherical and have equal sizes.
- The presence of outliers can influence the centroids, resulting in their movement from the central positions of the clusters.

2.5.1.2 DBSCAN Algorithm

The Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm is a density-based clustering method that distinguishes between low and high-point density regions. Two density parameters determine the allocation of points, which place them into one of three distinct categories. The presence of core and border points facilitates the formation of different clusters. There are significant challenges and difficulties in using the global density parameter in DBSCAN to detect clusters that reliably display significant differences in density [106]. The algorithm functions using two primary parameters:

- **Eps:** refers to the radius determining the maximum distance within which a designated number of points must be located to establish a high-density region.
- **Min_samples**: represents the least quantity of nearby points that must be present within the specified radius "eps" to establish a dense area.

2.5.1.3 OPTICS Algorithm

The Ordering Points Identify to Clustering Structure (OPTICS) algorithm is a density-based clustering method that does not necessitate the specification of the number of clusters as an input parameter. Random-shaped clusters can be effectively managed [107].

The OPTICS algorithm successfully addresses the challenges within the DBSCAN algorithm in the global density parameter by reliably detecting groups that display significant differences in density. The OPTICS approach operates similarly to an extended DBSCAN algorithm, accommodating an infinite number of distance parameters smaller than a global distance parameter, capable of being assigned an infinite value [106].

Limitation:

- Sensitivity Analysis of Parameters: Although the OPTICS algorithm does not necessitate a fixed radius parameter (eps) specification, it is still dependent on the min_samples and xi parameters. The selection of these parameters can substantially impact the outcomes of the clustering process., hence impacting the accuracy of cluster classification.

- Impact of High-Dimensional Data: As with many clustering algorithms, OPTICS' performance can degrade in high-dimensional spaces due to the curse of dimensionality. The interpretation of density changes becomes more complex, and distance-based computations might not accurately capture the actual density relationships.

- Cluster Shape and Density Variations: Although OPTICS has greater flexibility than specific other algorithms in its ability to handle diverse cluster shapes and densities, it may encounter difficulties when dealing with irregular cluster shapes or significant overlap in densities across different clusters.

2.5.1.4 GMM Algorithm

The Gaussian Mixture Model (GMM) algorithm is a distribution model-based approach that utilizes a unique approach to represent the dataset as a mixture of normal distributions. A GMM tends to cluster the data points from a singular distribution together. In contrast, The K-Means algorithm generates clusters that exhibit a spherical shape [107].

GMM algorithm can handle noise by redistributing it among the remaining clusters using the Gaussian distribution [108].

Limitation:

- Assumption of a Gaussian Distribution: The GMM assumes that the data points belonging to each cluster follow a Gaussian distribution; the GMM cannot effectively represent the data.

- Number of Components: Determining the most suitable number of components (clusters) is challenging. Poor selection of components may result in a problem of underfitting, while an excessive number of components might give rise to the issue of overfitting.

- Curse of Dimensionality: Similar to other clustering techniques, the performance of GMM in high-dimensional spaces results from increased data sparsity.

2.5.2 Supervised learning (SL)

Supervised learning is an approach to machine learning in which an algorithm can make predictions and evaluations by utilizing labeled training data. Supervised learning involves the utilization of an algorithm that is presented with a dataset that includes input data, which contains features, and the corresponding desired outputs, which encompass labels or target values. Supervised learning aims to acquire knowledge of the functional relationship between the input data and the intended outputs. Supervised learning problems can be categorized into two main groups: regression problems and classification problems. While unsupervised learning refers to a machine learning approach in which only input data is available, without any corresponding output variables [109].

Regression problems involve the task of predicting a continuous output variable. The algorithm learns knowledge of a function that establishes a correspondence between the input data and a contiguous set of values [110].

Classification problems involve predicting a discrete output variable or class label. The algorithm learns knowledge of a mathematical function that establishes a correspondence between the input data and a distinct category or class [110]. Table 2.1 provides an overview of the relevant research about the primary classification methods employed across various applications and datasets.

Table 2.1 Summary of Related Work for Classification Algorithms

Ref	year	Dataset	Algorithms	Application	Accuracy
[111]	2022	Flow Generation	DT SVM Naïve Bayes k-nearest neighbours (KNN) NN	Security	80.79% 78.78% 80.54% 81.57% 81.57%
[112]	2021	Flow Generation	k-NN SVM MLP DT NB	Traffic Classification	99.3% 88% 98.8% 89.8% 79%
[113]	2020	Flow Generation	SVM Naïve Bayes Nearest centroid	Traffic Classification	92.3% 96.8% 91%
[114]	2020	Kaggle	SVM (Linear) SVM (RBF) DT Random Forest K-NN	Traffic Classification	96.37% 70.40% 95.76% 94.92% 71.47%
[115]	2018	Universidad Del Cauca	Random forest Boosting with J48 LibSVM - linear kernel SMO - linear kernel J48 KNN with K = 23	Flow Detection	90.8 94.3 93.7 98 91.8 94.8

2.5.2.1 Decision Tree (DT) Algorithm

The decision tree is a supervised learning algorithm commonly employed for tasks involving classification and regression. The algorithm operates through a recursive process of dividing the dataset into subgroups, utilizing various features and corresponding values. At every internal node of the tree structure, a determination is made regarding the feature for splitting. Subsequently, the data is partitioned into branches based on this decision. The iterative procedure persists until a termination condition is satisfied, such as attaining a predetermined maximum depth or ensuring a minimum quantity of data points in every leaf node. The predicted class for new data points within a specific region in classification tasks is determined by the majority class present in each leaf node [109].

2.5.2.2 Naïve Bayes (NB) Algorithm

The Naïve Bayes (NB) algorithm is a probabilistic machine learning technique primarily employed for classification purposes. The Naïve Bayes classifier derives its foundation from Bayes' theorem and operates under the feature independence assumption, which accounts for including the term "naïve" in its naming. Despite making this simplifying assumption, Naïve Bayes frequently shows remarkable performance in practical applications, particularly in the context of text classification tasks. The calculation involves determining the likelihood of a data point being assigned to a particular class by considering the probabilities associated with its features given that class. The prediction is determined by assigning the class with the highest probability [116].

2.5.2.3 Support Vector Machine (SVM) Algorithm

The Support Vector Machine (SVM) is a robust supervised learning algorithm for classification and regression in various domains. In classification, Support Vector Machines (SVM) are utilized to identify the most optimal hyperplane that effectively separates data points belonging to distinct classes within a space characterized by many dimensions. The primary goal is to optimize the margin, which refers to the distance between the nearest data points belonging to distinct classes, known as support vectors. Support Vector Machines (SVMs) can handle data that is not linearly separable. By using kernel functions, which transform the data into a higher-dimensional space, it is possible. In this transformed space, the SVM can then separate the data points [117].

2.5.2.4 K-Nearest Neighbors (KNN) Algorithm

The K-Nearest Neighbors (KNN) algorithm is an easy technique in supervised learning commonly employed for classification and regression purposes. The K-Nearest Neighbors (KNN) algorithm predicts the class or value of a data point by considering the majority class or average value of its K nearest neighbors in the feature space. The selection of the value of K is a crucial hyperparameter. Lower values of K can lead to more adaptable decision boundaries, allowing for greater flexibility. However, this increased flexibility also makes them more vulnerable to the influence of noise in the data. On the other hand, higher values of K can produce smoother decision boundaries, but this smoothness may cause them to overlook local patterns in the data [118].

2.5.2.5 Random Forest (RF) Algorithm

The Random Forest algorithm is a supervised learning technique that integrates multiple decision trees to enhance the accuracy and reliability of predictions for classification and regression problems. The algorithm operates by generating numerous decision trees using random subsets of the dataset and random subsets of features. During the process of prediction, each tree generates its prediction. In classification tasks, the ultimate prediction is determined by aggregating the individual trees' predictions through a majority voting mechanism. The utilization of Random Forest has been shown to mitigate the issue of overfitting and enhance the overall generalization performance of a model compared to using a single decision tree [119].

2.6 Evaluation Measures

This section provides an overview of the different methods used to assess the effectiveness of machine learning models. Developing a comprehensive understanding of the performance of models on unseen data is crucial to making informed decisions regarding model selection.

2.6.1 Validation Set

The accessible data was partitioned into three sets: a training set, a validation set, and a test set. The training set was utilized to fine-tune the model parameters, while the validation set was employed to adjust the hyperparameters, prevent overfitting, and enhance the models' ability to generalize. Then, the test set assessed the models' generalizability on unseen data. A frequently employed division entails allocating 70% of the dataset for training, 15% for validation, and 15% for testing. However, it should be noted that these percentages may change depending on the size of the dataset and the specifics of the problem [120]. The purpose of employing a validation set is to assess the efficacy of a model and fine-tune its hyperparameters. Let us assume that the value of val_size is:

val_size = 0.15 **(2.1)**

If the model displays high performance on the training set but poor results on the validation set, it indicates potential overfitting. In instances of overfitting, it is necessary to adjust the hyperparameters to deal with and fix the issue effectively.

2.6.2 Performance Metrics

The evaluation of classification predictor models involves using various measures, including precision, recall, F1 score, and accuracy. These measures can be calculated based on Table 2.2 and the equations below. The Confusion Matrix (CM) is a crucial method, as all the measures depend entirely on the CM.

Table 2.2. Confusion Matrix

		Predicted Class	
		+	-
Actual Class	+	**TP**	**FN**
	_	**FP**	**TN**

The concept of CM refers to evaluating the efficacy of a supervised learning technique. The confusion matrix (CM) is presented in Table 2.2, which displays the number of true positive (TP) instances. TP refers to the accurate prediction of records belonging to class zero as class zero. The true negative (TN) refers to the count of instances where records belonging to class one are accurately predicted as class one. A false positive (FP) refers to the count of instances where records corresponding to class one are erroneously predicted as class zero. The term "False Negative" (FN) refers to the count of instances where records belonging to class zero are incorrectly classified as class one [121]. To calculate performance metrics can use equations 2.2 to 2.5.

$$\textbf{\textit{Precision}} = \mathbf{TP/(TP+FP)} \qquad \mathbf{(2.2)}$$

$$\textbf{\textit{Recall}} = \mathbf{TP/(TP+FN)} \qquad \mathbf{(2.3)}$$

$$\textbf{\textit{F1}}-\textbf{\textit{Measure}} = \mathbf{2 * (precision * recall) / (precision + recall)} \qquad \mathbf{(2.4)}$$

$$\textbf{Accuracy} = \mathbf{(TP+TN)/(TP+TN+FP+FN\ )} \qquad \mathbf{(2.5)}$$

2.6.3 Silhouette Score

The silhouette score is a metric that quantifies the degree of similarity between an object and its cluster, reflecting the cohesion within the cluster and the isolation from other clusters. The silhouette value is a numerical measure within the range of [1, -1]. A higher silhouette value signifies a strong correspondence between the object and its assigned cluster while indicating a weaker correspondence with neighboring clusters [122]. The calculation of this value can be performed using equations 2.6 and 2.7.

Use the following formula to determine the silhouette coefficient of s(x)

$$\mathbf{s(x) = (b(x) - a(x)) / max(a(x), b(x))} \qquad \mathbf{(2.6)}$$

Calculate the overall data's average silhouette coefficient of S:

$$\mathbf{S = (s(1) + s(2) + ... + s(n)) / n.} \qquad \mathbf{(2.7)}$$

Where n represents data points, x represents data points within the same cluster, a(x) is the average distance between many data points in the same cluster, and b(x) is the average distance between data points in the closest neighboring cluster.

2.7 Queue Scheduling

Packet scheduling plays a crucial role in various networks, including SDN. Its significance consists of efficiently managing and prioritizing data packets before they are transmitted, ultimately leading to optimized network performance. By carefully organizing the order in which packets are processed and sent, packet scheduling aims to enhance the overall efficiency and quality of network communication. [123].

Examining traffic management in an SDN environment involves exploring three distinct approaches: heuristic, parametric, and model-based. These approaches aim to improve the SDN controller's and OpenFlow switch's performance by implementing priority queueing mechanisms [124].

- Heuristic Approaches refer to algorithms that utilize rules-of-thumb and practical techniques to identify and control queue traffic models without considering any specific model, for instance, using a heuristic approach for achieving load balancing among queues. Or, the proposed method utilizes a Priority Queue (PQ) to manage incoming data traffic. It entails enqueuing packets into the appropriate queue based on their priority, with the high priority queue being dequeued first. Alternatively, a Multilevel Queue (MLQ) with distinct priority classes can be employed [124].

- Parametric Approaches refer to strategies in which the management of queues relies on fewer heuristics and places a greater emphasis on methodological solutions. In an improved manner, the QoS of an SDN is characterized by one or more parameters, which are then utilized in the optimization process. The parameters mentioned above define the static models used for this optimization. For instance, one can employ methodologies to simulate and regulate queuing delays by considering specific network parameters. Alternatively, can utilize Quality of Experience (QoE) or Mean

Opinion Score (MOS) metrics to evaluate the perceived quality of voice and audio in telecommunication systems [124].

- Model-Based Approaches involve the consideration of a mathematical model that represents the movement of packets within a queue. For instance, using the approach involves applying queuing theory, which considers various factors such as the flow table size, the rate at which packets arrive, and other relevant considerations. Alternatively, utilizing particular queue models, like M/M/1, M/M/C, M/M/k, and M/G/k, which fit into particular distributions like the Poisson distribution and the exponential distribution, is possible [124].

In the environment of SDN, utilizing a priority queue and machine learning techniques together can enhance the network's responsiveness and reliability. Furthermore, this approach assists in the reduction of network congestion, the minimization of latency, and the optimization of the utilization of existing network resources. The primary objective is to optimize packet loss reduction and save bandwidth, specifically to enhance the Quality of Service [124].

Single-Queue as a priority queue is the scheduling mechanism of the queue that displays a significant occurrence of starvation due to the arrival of packets in the ready queue being prioritized only based on their size, type, and priority. For instance, in the initial stage of the process, only high-priority tasks are executed, while low-priority tasks must wait until the high-priority tasks are completed. This situation leads to starvation for low-priority tasks and other priorities. To handle a starvation issue, a possible approach involves implementing a multilevel queue (MLQ) model and using a threshold concept to prevent the occurrence of priority-based starvation [123].

The range of priority values from 0 to 127 is not universally fixed or constant across all systems. While some systems may limit the priority range to 0–15, 16–31, or 0-99 and 100–139 for practical reasons, others might utilize the entire range [125, 126].

The specific range of priority values a system supports is typically determined by its underlying architecture, operating system, and scheduling algorithm and model. It can vary depending on the design choices made by the system developers.

2.8 Parameters Calculation

This section will explain the methods used for the computation of significant parameters.

2.8.1 Response Time (RT)

The calculation can be performed by subtracting the Start Time from the End Time of a specific request or process [127]. In the field of networking and application performance analysis, response time can be generally deconstructed into two primary constituents [128]:

Network latency (NL) refers to the duration required for data packets to move from the origin to the target location and return. The architecture includes various elements, such as propagation delay, transmission delay, processing delay, queuing delay, and serialization delay. Network latency refers to the duration data travels and returns to the sender.

Service Time (ST) refers to the duration required for a server or network device to handle an incoming request and generate a corresponding response. The server works in various operations, such as computations, data retrieval, and other tasks.

The formula for calculating Response Time (RT) is:

$$\mathbf{RT = Et - St} \qquad \mathbf{(2.8)}$$

Where Et is End_Time , and St is Start_Time

Alternatively, can express it as:

$$\mathbf{RT = NL + ST} \qquad \mathbf{(2.9)}$$

Where NL is Network_Latency, and ST is Service_Time

Both formulas are considered to be comparable and provide the overall duration required for the request to be processed and responded to, accounting for both the time spent in transmitting data over the network (referred to as Network Latency) and the time spent in processing the request on the server or network device (known as Service Time).

Lower response times enhance network efficiency, application performance, and QoS [129]. It necessitates the optimization of both network latency and service time.

2.8.2 Latency (L)

It refers to the duration required for data packets to move from the origin to the target location and return. The architecture includes various elements, such as propagation delay, transmission delay, processing delay, and queuing delay [130].

Service Time (ST) can be calculated from the duration required by the server to handle the request and produce the corresponding response. The value is accessible from the elapsed attribute of the response object that the requests returned. get() function.

Based on Eq. 2.8

The formula to calculate Latency is:

$$\textbf{Latency} = \textbf{RT} - \textbf{ST} \qquad \textbf{(2.10)}$$

Where RT is response time, and ST is Service Time, then

Latency = (Network Latency + Service Time) - Service Time

$$\textbf{Latency} = \textbf{Network Latency} \qquad \textbf{(2.11)}$$

2.8.3 Throughput

Quantitatively measured by the number of requests handled per unit of time, typically expressed as requests per second. The objective is to determine the duration in seconds by considering the average time required to complete all the requests [131].

The formula to calculate Throughput is:

Throughput = 1 / (Average_RT * Requests) **(2.12)**

Where:

The variable Average_RT represents the mean value of the response time collected within a designated time.

Requests are the total number of requests made within a given time interval.

2.8.4 Degree of Load Balancing

The load balancing degree is a valuable metric for evaluating the efficacy of workload distribution across resources within a system or load balancer. The load balancing degree can be determined by utilizing the Coefficient of Variation [132], which can be calculated by considering the Mean Response Time, Variance, and Standard Deviation of Response Times [133].

The Mean Response Time (Mean_RT) is determined by computing the average Response Time (RT) through the division of the average value of all individual Response Times by the total count of Response Times (len(RT)).

Mean_RT = sum(RT) / len(RT) **(2.13)**

The Sum of Squares (SS) is calculated by summing the squared deviations between each Response Time (rt) and the Mean Response Time (Mean_RT).

SS = sum((rt - Mean_RT) ** 2 for rt in RT) **(2.14)**

Variance is a statistical metric that quantifies the degree to which Response Times show variability or deviation from the Mean Response Time. The calculation involves the division of the Sum of Squares (SS) by the total number of Response Times (len (RT)).

Variance = SS / len(RT) **(2.15)**

The Standard Deviation is a statistical measure derived from the variance by taking its square root. It serves as an indicator of the distribution or level of variability in the response times.

Std_Deviation = sqrt(Variance) **(2.16)**

Coefficient of Variation (CV) is a metric used to measure the effectiveness or performance of a system or process. The concept refers to a value obtained by dividing the variance by the standard deviation. The metric provides an objective evaluation of the performance of Load Balancing.

Coefficient of Variation (CV) = Variance / Std_Deviation (2.17)

The load balancing degree is determined by subtracting the coefficient of variation from 1. This metric represents the level of load balancing, with a higher numerical value indicating a higher degree of load balancing and improved performance.

load_balancing_degree = 1 – CV (2.18)

2.9 The Research Dataset

In this section, two distinct file classifications are employed to generate data and create datasets. The initial group, called Type 1, includes non-real files that produce dynamic data by manual methods and consists of five files. On the other hand, the second category, referred to as type 2, comprises real files sourced from the University of California and collected from Facebook [134] , consisting of 16 files. Table 2.3 provides a comprehensive investigation of the data and files utilized.

Table 2.3 Types of Data and Files

Requests	Source	Details	No. Files	Types
Type 1	Flow Generation by Creating Files Manually	Creating five files of different sizes (1M, 10, 25M, 50M, and 75M)	5	1.txt, 10.txt, 25.txt, 50.txt, and 75.txt
Type 2	Real Files (Facebook) - University of California, San Diego [134]	18% of the data was less than 0.1 Kbyte (3 files), 19%–38% was between 0.1 and 10 Kbyte (3 files), 39%–89% was 1 Mbyte (8 files), and 90%–100% was less than 10 Mbyte (2 files).	16	cache1.1, cache1.2, cache1.3,cache2.1, cache2.2, cache2.3, cache3.1, cache3.2, cache3.3, cache3.4, cache3.5, cache3.6, cache3.7, cache3.8, cache4.1 and cache4.2

Chapter Three

The Proposed Model

3.1 Overview

The dissertation proposed a new approach to load balancing in SDN networks by proposing an integrated three-model algorithm, KNN-MLQLRL, which combines Machine Learning with Multilevel Queue and Load Balancing Scheduling (ML-MLQLBS). This integration aims to create an intelligent dynamic load balancing model, thus enhancing the network's Quality of Service (QoS) and workload distribution.

The proposed model comprises seven different stages. The initial phase encompasses generating, capturing, analyzing, and collecting packets to create a dataset, and the preprocessing stage is the second stage. Then, clustering methods are employed in the third stage to assign class labels. The fourth stage employs Machine Learning Classification (MLC), the K-Nearest Neighbors (KNN) algorithm, to classify and predict priorities effectively. The fifth stage, on the other hand, uses the suggested Multilevel Queuing (MLQ) method to get priorities from the MLC and store them before sending them to servers. The proposed Load Balancing Scheduling (LBS) method is implemented in the sixth stage. This method uses a new algorithm called least resource load (LRL) to determine which server has the least CPU and memory. Finally, the seventh stage is designing the proposed model. This model includes the results from the previous stages and intelligent dynamic load balancing to distribute work among servers better.

3.2 Diagram of Proposed Model

The design and development of the model depend on integrating three algorithms. The initial algorithm employs machine learning methodologies, namely the K-Nearest Neighbors (KNN) algorithm, to classify and predict priorities effectively. The second approach employs queue scheduling by utilizing the suggested multilevel queue model (MLQ) to store priorities before their transmission to servers. The third type of load balancing scheduling (LBS) involves the utilization of a proposed algorithm called least resource load (LRL).

The proposed approach involves the integration of three algorithms, namely Machine Learning with Multilevel Queue and Load Balancing Scheduling (ML-MLQLRL). This integration aims to develop an intelligent model called KNN-MLQLRL, which facilitates intelligent dynamic load balancing. Figure 3.1 represents the diagram of the proposed model, which highlights its position inside the SDN network architecture. This model is positioned in the second layer of the control plane, located within the SDN controller.

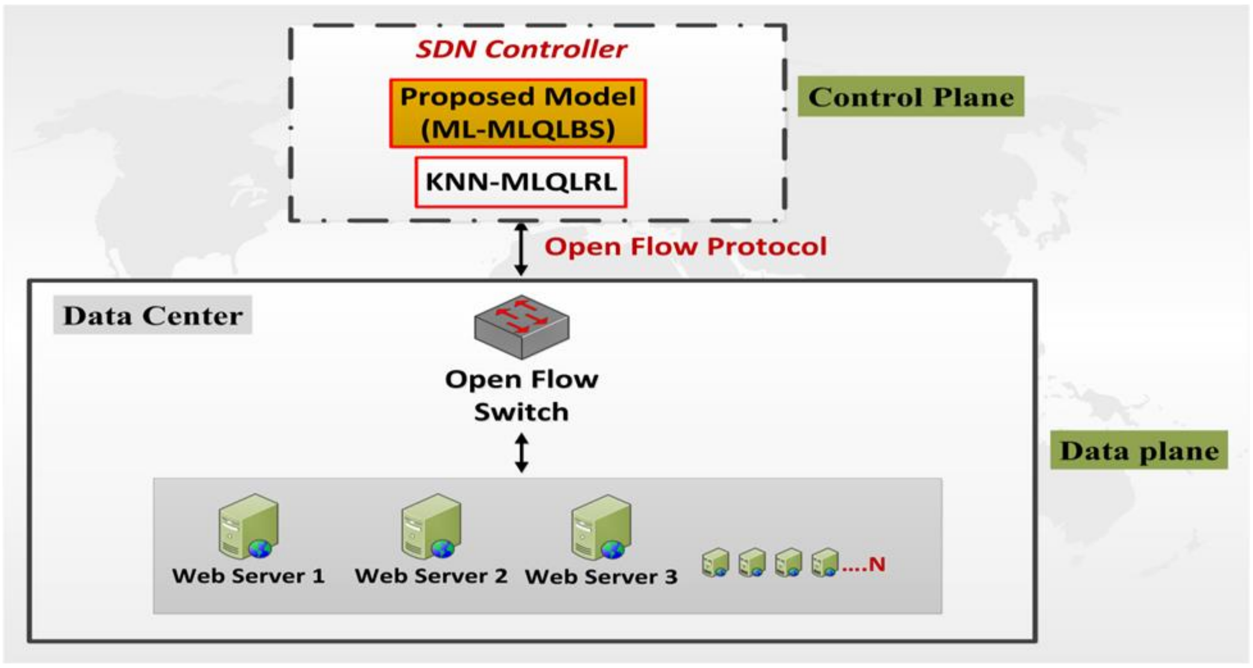


Figure 3.1 Proposed Model Diagram

Diagram 3.2 illustrates the general structure of the suggested model, encompassing the six previously explained stages and the last stage dedicated to model design. The first stage is the generation and collection of data, and the following phase is dataset reprocessing. The third stage involves creating clustering (labeling) for the dataset, while the fourth stage entails training the data and performing classification using machine learning classification algorithms. Specifically, the K-Nearest Neighbors (KNN) algorithm is employed to construct a predictive model. Upon receiving requests, the model initiates its operations. The initial step involves making predictions to identify the clustering type and priorities and sending them to the next stage.

Subsequently, the fifth phase includes building a queueing model; his work receives the priorities from the classification model, which consists of five queues. The first queue is designated for high priorities, while the last is designated for low priorities. The distribution of priorities is dependent on the threshold value assigned to each queue. Lastly, the sixth stage involves employing the proposed LRL algorithm to compute the resource capacity. Subsequently, the priorities are transmitted to the server with the least energy use.

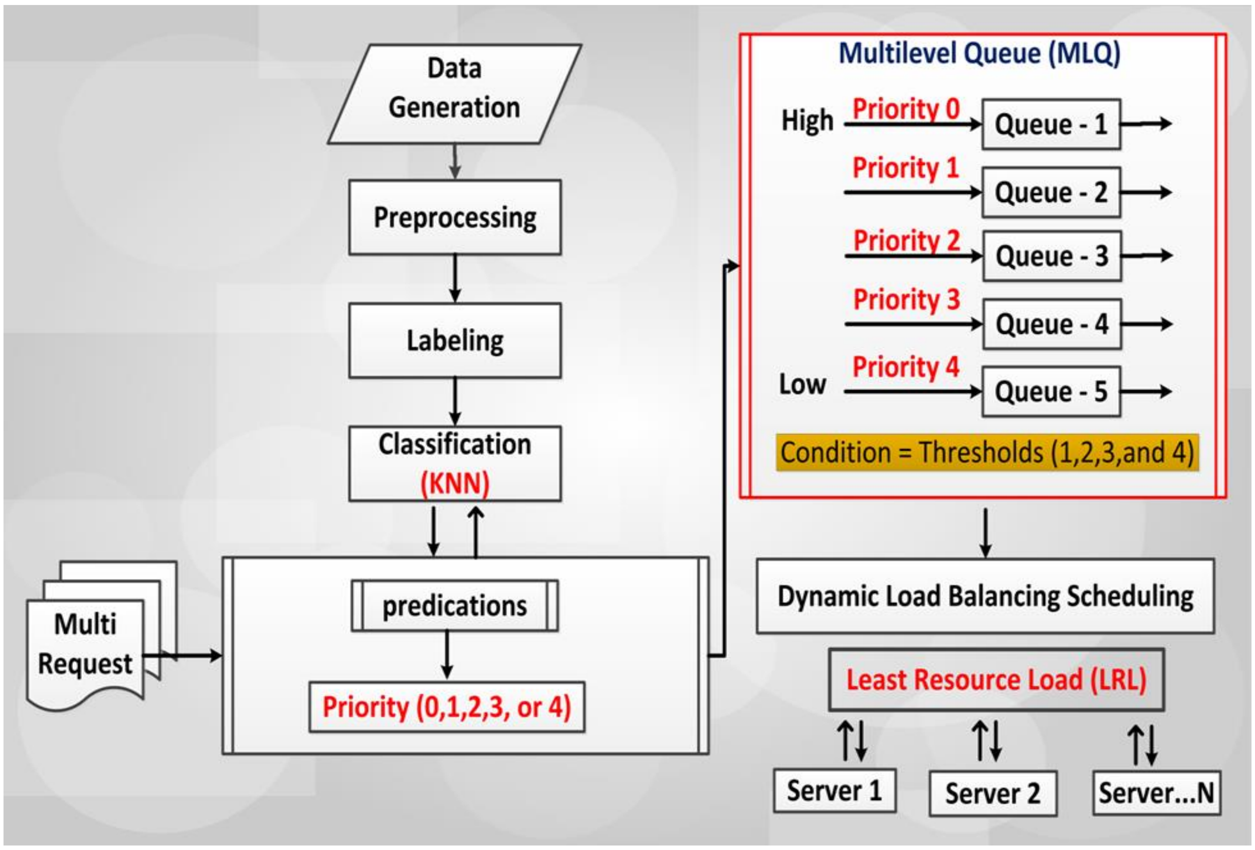


Figure 3.2: Proposed Model Structure

3.3 Framework of Proposed Model

The framework presented in this model is characterized by a critical development process consisting of seven distinct stages. Figure 3.3 depicts each process stage as crucial in determining the design model and functionality. In the following sections, this dissertation will meticulously explore the intricate aspects of these stages, thereby shedding light on their individual and collective importance. This dissertation aims to comprehensively understand how these stages enhance the proposed model's efficacy and robustness.

The first stage is data generation, encompassing request generation, traffic capturing, data collection, and dataset creation. The second stage involves preprocessing the dataset, which includes tasks such as data cleaning, feature selection, and data scaling.

The third stage encompasses Unsupervised Machine Learning techniques, which comprise Density-based algorithms like Optics, Distribution-based algorithms such as the GMM, and newly proposed algorithms derived from Optics and GMM.

The four stages are the classification algorithms consisting of six specific algorithms, namely Decision Trees (DT), Support Vector Machines (SVM), Random Forest (RF), Artificial Neural Networks (ANN), k-Nearest Neighbors (k-NN), and Naive Bayes. The selection of the optimal algorithm is determined by calculating the accuracy of the training, validation, and test sets. Additionally, the confusion matrix and evaluation metrics are utilized in this process.

The five stages encompass Queue Scheduling, which comprises a proposed multi-level queue and an approach that contributed to solving the issue of priority starvation. The sixth stage involves utilizing a load-balancing scheduling mechanism encompassing six distinct algorithms. Four algorithms are employed for load balancing scheduling (LBS) within the SDN network. These four algorithms are R, RR, WRR, and Least Connection. Additionally, two dynamic algorithms have been devised. The first algorithm, Least Resource Load (LRL), calculates the CPU and memory to determine load balancing. The second algorithm, the Hybrid from LC and LRL (LCLRL), combines the principles of least connection and least resource load (CPU + Memory) to achieve load balancing.

Finally, the seventh stage is the proposed model, known as the KNN-MLQLRL, based on utilizing the algorithm LBS (also referred to as LRL). This model incorporates the linking of two algorithms, Machine Learning-Multi-Level Queue, and is designed as the final model, ML-MLQLBS.

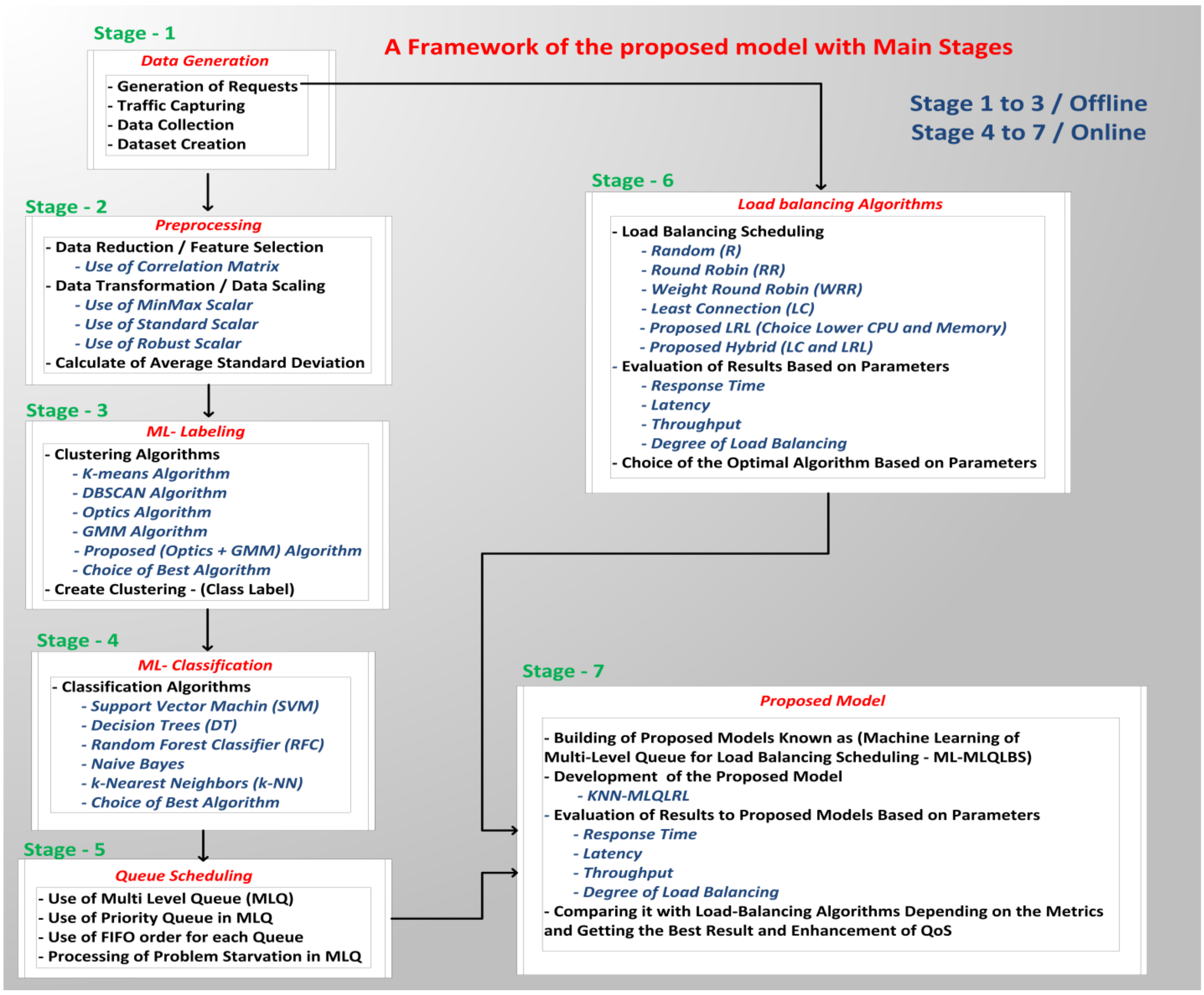

A Framework of the proposed model with Main Stages
Stage 1 to 3 / Offline
Stage 4 to 7 / Online
Stage - 1
Data Generation
- Generation of Requests
- Traffic Capturing
- Data Collection
- Dataset Creation
Stage - 2
Preprocessing
- Data Reduction / Feature Selection
- Use of Correlation Matrix
- Data Transformation / Data Scaling
- Use of MinMax Scalar
- Use of Standard Scalar
- Use of Robust Scalar
- Calculate of Average Standard Deviation
Stage - 3
ML- Labeling
- Clustering Algorithms
- K-means Algorithm
- DBSCAN Algorithm
- Optics Algorithm
- GMM Algorithm
- Proposed (Optics + GMM) Algorithm
- Choice of Best Algorithm
- Create Clustering - (Class Label)
Stage - 4
ML- Classification
- Classification Algorithms
- Support Vector Machin (SVM)
- Decision Trees (DT)
- Random Forest Classifier (RFC)
- Naive Bayes
- k-Nearest Neighbors (k-NN)
- Choice of Best Algorithm
Stage - 5
Queue Scheduling
- Use of Multi Level Queue (MLQ)
- Use of Priority Queue in MLQ
- Use of FIFO order for each Queue
- Processing of Problem Starvation in MLQ
Stage - 6
Load balancing Algorithms
- Load Balancing Scheduling
- Random (R)
- Round Robin (RR)
- Weight Round Robin (WRR)
- Least Connection (LC)
- Proposed LRL (Choice Lower CPU and Memory)
- Proposed Hybrid (LC and LRL)
- Evaluation of Results Based on Parameters
- Response Time
- Latency
- Throughput
- Degree of Load Balancing
- Choice of the Optimal Algorithm Based on Parameters
Stage - 7
Proposed Model
- Building of Proposed Models Known as (Machine Learning of Multi-Level Queue for Load Balancing Scheduling - ML-MLQLBS)
- Development of the Proposed Model
- KNN-MLQLRL
- Evaluation of Results to Proposed Models Based on Parameters
- Response Time
- Latency
- Throughput
- Degree of Load Balancing
- Comparing it with Load-Balancing Algorithms Depending on the Metrics and Getting the Best Result and Enhancement of QoS

Figure 3.3 Framework of Model Diagram

3.3.1 Stage 1 - The Data Generation

This stage explains the process of generating the datasets utilized in the dissertation. As shown in Chapter 2, Section 2.9, and Table 2.3, Two distinct kinds of files will generate two different datasets. The current stage is classified into four distinct segments, each with a designated purpose.

3.3.1.1 Request Generation

Requests can be classified into two types for file transfer: static and dynamic,as stated in Chapter 2, specifically in Section (2.3.1). The dissertation will employ static files to generate data and construct datasets based on two distinct file classifications, as mentioned in Chapter 2, Section 2.9. Additionally, static files will be utilized for traffic generation to facilitate load balancing scheduling. The methodology for generating requests employing the Mininet emulator and SDN architecture._While dynamic files will be utilized for traffic generation to facilitate the proposed model of intelligent dynamic load balancing.

3.3.1.2 Traffic Capturing

This dissertation will employ the second categorization, which involves utilizing the statistical strategy to capture data. Specifically, the Wireshark program will be installed within the Linux to capture the requests for both types (requests for type 1 and type 2)

3.3.1.3 Data Collection

The Wireshark program and the statistics tab provide access to various attributes. Then, it chooses the TCP tab to collect the data, such as traffic patterns, protocols, packet sizes, IPs, and other relevant statistical measures. Data were collected for both categories (requests for type 1 and type 2).

3.3.1.4 Dataset Creation

At this step, the dataset will be generated after previously compiled, including 15 different features. Two distinct datasets will be generated based on requests for types 1 and 2. Aggregating requests from type 1, which had 4184 rows and

15 distinct attributes, created the initial data set. Similarly, the second dataset was generated by aggregating requests from type 2, which contained 15 attributes but had a larger sample size of 30,000 rows. Table 3.1 comprehensively explains the various characteristics associated with each feature.

Table 3.1 Dataset Features

No.	Feature	Details
1	S	Number of requests
2	Address A	Represent the IP address of the source's network node, such as a device or server.
3	Port A	Represent the source network node's communication port number. TCP ports identify apps and services on a device.
4	Address B	Represent the IP address of the destination network node receiving the communication.
5	Port B	Represent the destination network node's communication port number.
6	Packets	Represent displays how many packets the source (A) and destination (B) exchanged during the communication session. Data is sent in packets.
7	Bytes	Represent the total number of bytes sent between source (A) and destination (B) during the communication session.
8	Packets A to B	The number of packets sent from source (A) to destination (B) indicates the quantity or volume of data transmitted.
9	Bytes A to B	Represent the total number of bytes sent from source (A) to destination (B), representing the size or amount of data transmitted.
10	Packets B to A	The number of packets sent from the destination (B) to source (A) indicates the quantity or volume of data transmitted.
11	Bytes B to A	Represent the total number of bytes sent from the destination (B) to source (A), indicating the size or amount of data transmitted.
12	Rel Start	Refers to the relative start time of the communication session
13	Duration	Represent of session duration between the source (A) and destination (B) communication, usually in seconds.
14	Bits/s A to B	Represent the average data transfer rate from source (A) to destination (B) in bits per second. Indicate to average data transfer speed.
15	Bits/s B to A	Represent the average data transfer rate from destination (B) to source (A) in bits per second. Indicate to average data transfer speed.

3.3.2 Stage 2 - The Preprocessing

The current phase is essential to the data analysis process since it includes converting raw data into carefully structured datasets. According to Chapter 2, Section 2.4, the dataset requires the implementation of data planning processes to reduce irrelevant and redundant data.

Data preprocessing is an essential phase in the field of data analytics. The current stage comprises two significant steps: data reduction, and data transformation. Preprocessing will apply to two datasets in each step.

3.3.2.1 Data Reduction

Based on the information provided in Chapter 2, Section 2.4.2, various methodologies exist for the data reduction procedure. This dissertation employs a feature selection technique that utilizes the correlation matrix approach to compute the degree of correlation between columns (features). The resulting correlation values range from 0 to 1, with a higher value indicating a more substantial degree of correlation and a lower value indicating a weaker degree of correlation.

After implementing the feature selection technique on datasets 1 and 2, the features are reduced from 15 to 6. The process involves selecting a limited set of 6 features that exhibit high values, focusing on features near a value of 1. Table 3.2 explains the datasets. In this scenario, a correlation value closer to 1 indicates a higher correlation, which is selected for datasets. Conversely, a correlation value of 0 signifies a lack of correlation, requiring removal from the dataset.

Table 3.2 Datasets Dimension after using Feature Selection

Dataset	Dimension	Details
Dataset 1	7 Features * 4184 rows	1. Packets 2. Bytes 3. Packets A to B 4. Bytes A to B 5. Packets B to A 6. Bytes B to A
Dataset 2	7 Features * 30,000 rows	

3.3.2.2 Data Transformation

Data transformation using data scaling is a widely employed technique in data preparation. Its primary objective is to convert the dataset's characteristic values to a standardized scale. The significance of scaling is based on its ability to assure equitable treatment of features with varying sizes or magnitudes by machine learning algorithms. Additionally, it can enhance the convergence of specific optimization techniques. This dissertation employs three fundamental strategies and ultimately selects the optimal technique by the standard deviation is generally employed to measure the dispersion or diversity of data points within a singular feature and select the most suitable dataset based on the result to average lower standard deviations across all features. At this stage, the scaling data will be computed using three strategies.

1. The MinMaxScaler is a data preprocessing technique that employs normalization to scale the values inside a dataset. It ensures that the minimum value is transformed to 0 while the maximum value is transformed to 1.

2. The StandardScaler is a data preprocessing technique that applies standardization to a dataset, resulting in a mean of 0 and a standard deviation of 1.

3. The RobustScaler is another data preprocessing technique that scales the data to a range between -3 and 3, making it more robust against outliers.

3.3.3 Stage 3 - Unsupervised Machine Learning (UML)

According to the importance presented in Chapter 2, Section 2.5.1, unsupervised algorithms work with unlabeled data to discover patterns, structures, or relationships within the data and then assign labels to the dataset. The dissertation proposes an algorithm by combining Optics and GMM and comparing it with 1) partitioning algorithms (K-means), 2) density-based algorithms (DBSCAN and Optics), and 3) distribution-based algorithms (GMM).

3.3.3.1 Proposed (OPTICS + GMM) Algorithm

The proposal involves combining two algorithms, OPTICS and GMM, which effectively identify clusters with diverse shapes and handle datasets that incorporate noise. In the first algorithm, OPTICS excels by adeptly partitioning data points into distinct clusters while designating points with a reachability distance of -1 as noise within separate, distinct noise clusters. However, this approach can be challenging as the data points labeled as noise remain separate from other clusters, necessitating an intervention to handle this problem.

The second algorithm uses a probabilistic distribution that characterizes data points within each cluster to address this. The dissertation proposes this algorithm as a solution to the issue of data points labeled as noise. It achieves this by reassigning these data points to the existing true clusters initially established by the first algorithm. This approach refrains from creating new clusters and ensures that the noise-labeled data points become integrated within the appropriate existing clusters. This sequential approach reduces the isolation of noise points and introduces both algorithms' strengths to yield improved clustering results.

Algorithm 3.1 explains the phases of the suggested optimization algorithm to apply clustering to a datasets.

Algorithm 3.1: Proposed Combined OPTICS and GMM Algorithms

```
Input:
DS: load dataset(Packets, Bytes, Packets A to B, Bytes A to B, Packets B to A, Bytes B to A,
Bits/s A to B, Bits/s B to A)
Ms: Min_samples
Xi: Minimum distance separation between clusters
Mc: Min_cluster_size
Nc: Number of Components
CLG: cluster_labels_gmm
Output:
N_c: Number of Clusters
NL: Noise Label (-1)
N_c_new: Number of Clusters after using GMM
N_fc: Number of Final Clusters
Begin
Load the dataset
DS = load_dataset()
Initialize variables
N_c = 0,  NL = -1, N_c_new = [], N_fc = []
  while True:
      Apply OPTICS Algorithm
      model, cluster_labels_optics, noise_labels, _ = apply_optics(DS, Ms, Xi, Mc)
      Partition data into clusters (N_c)
      N_c = max(cluster_labels_optics) + 1
      Designate noise points with reachability distance of -1 (NL)
      NL = noise_labels
      Apply GMM Algorithm
      gmm_model.fit(DS[NL == -1])
      CLG = gmm_model.predict(DS[NL == -1])
      Assign the GMM cluster labels to the noise-labeled data points (NL)
      N_c_new = cluster_labels_optics.copy()
      N_c_new[NL == -1] = CLG
      Creating a number of final clusters (N_fc)
      N_fc = [cluster for cluster in N_c_new if cluster != -1]
      Check convergence condition
      if len(set(N_fc)) == N_c:
          break
  return N_c, NL, N_c_new, N_fc
```

3.3.4 Stage 4 - Supervised Machine learning (SML)

The dissertation employs classification techniques to handle dataset classification and prediction, as mentioned in Chapter 2, Section 2.5.2. It employs five representative classifiers, namely: (1) decision tree (DT), (2) support vector machine (SVM), (3) k-nearest neighbour, (4) naïve Bayes (NB), and (5) random forest (RF. This decision is based on the Algorithm's excellent performance in classification tasks, as shown by achieving high evaluation metrics such as accuracy and performance metrics. The dissertation ultimately decides to utilize the K-nearest neighbour algorithm as a part of the objective function in all experiments.

3.3.5 Stage 5 - Proposed Queue Scheduling

The idea suggests the utilization of queue scheduling as an achievable approach for effectively managing and storing priorities coming from the classification and prediction model. This strategy improves networks' performance and Quality of Service (QoS) by reducing packet loss and efficiently managing priorities based on predictions. Before transmitting data to the servers, it is crucial to construct a queue to store various essential priorities.

As described in Chapter 2, Section 2.7, examining traffic management in the SDN framework encompasses exploring three distinct methodologies: heuristic, parametric, and model-based. This dissertation employs a heuristic technique to achieve load balancing among priorities through single or multiple queues. This approach guarantees efficient distribution of workloads and handles the issue of packet starvation.

To deal with the problem of packet starvation, the researcher suggests the utilization of a Multilevel Queue (MLQ) as an alternative to a singular queue. This strategy enhances priority management, improving network performance and quality of service (QoS).

The following steps illustrate the Multi-Level Queue (MLQ) model that has been proposed.

- The quantity of queues depends on the number of clusters or priorities previously established during the third stage through unsupervised approaches. Hence, the proposed model utilizes the Multi-Level Queue (MLQ) approach, wherein five distinct queues are established for the five priority levels, namely 0, 1, 2, 3, and 4.

- The approach suggests utilizing the priority concept for distributing between MLQs. Specifically, a priority of 0 will be inserted into queue 0, a priority of 1 will be stored in queue 1, and a priority of 4 will be stored in queue 4.

-The proposed strategy involves implementing a First-In-First-Out (FIFO) order to manage the storage of incoming priorities within each queue.

- Within the concept of prioritization, the transmission of tasks follows a hierarchical structure. The initial task, denoted as priority 0, is consistently assigned the highest priority and is dealt with first. Subsequently, upon completing all tasks assigned with priority 0, tasks with priority 1 are transmitted. This sequential pattern continues until all tasks are handled based on their priorities. The scenario mentioned above causes an issue for lower priorities, specifically those ranked as priorities 3 and 4, commonly referred to as starvation. This issue results in delays in transmitting low-priority data and an overall decrease in network performance. This dissertation proposes solving the starving issue by employing the threshold concept. In this context, a threshold value is established for each priority level, and once the specified value has been reached, the lower-priority tasks will be process. The important phase of the MLQ model is demonstrated by Algorithm (3.2)

Algorithm 3.2: Proposed Multi-level Queue (MLQ) Model

```
Input
Initialize Priority queue (0, 1, 2, 3, and 4)
Counter = 0
threshold1 = 8
threshold2 = 9
threshold3 = 10
threshold4 = 11
Push()
Output:
Pop()
Begin
1. queues = {0: [], 1: [], 2: [], 3: [], 4: []}
2. Setting a counter and thresholds
3. Call the push()
  packet = get_next_packet()
  priority = get_packet_priority(packet)
  push(packet, priority)
4. Call the pop()
  popped_packet = pop(queues)
  process_popped_packet(popped_packet)
5. while True:
        Call the push()
        Call the pop()
        counter += 1
        Check the status of the thresholds
        if counter == threshold1 and queues[1]:
            return 1
        elif counter == threshold2 and queues[2]:
            return 2
        elif counter == threshold3 and queues[3]:
            return 3
        elif counter == threshold4 and queues[4]:
            return 4
        elif counter == threshold4 and not any(queues.values()):
            counter = 0
        Check priority queues
       for priority in range(5):
            if queues[priority]:
                return priority
        Check if packet is empty
        if not any(queues.values()):
            return None
```

The input for Algorithm 3.2 is the creation of five queues, assigning values to thresholds, initializing a counter with a value of zero and using the push function to store priorities. The algorithm's output involves using the pop function to assign priority to servers.

- The initial two steps involve creating queues and managing a counter and thresholds inside a queue model.
- Steps 3 and 4 involve utilizing the push and pop operations on the priority.
- Step 5 involve recall of the push and pop operations on the priority involves performing operations and then increment the counter's value for each priority if the pop function is utilized. Then assesses whether the counter reached specific predetermined thresholds and whether particular conditions relevant to the queue were achieved. The provided conditions, such as counter = threshold1 and queue [1], counter = threshold2 and queue [2], etc., suggest that the counter's value is being compared to various thresholds while considering the queue's state at specified indices. If a match is found, a value is delivered according to the priority type, and the process returns to step 5 to invoke other priorities. The final instruction, "counter = 0," serves to reset the counter to its initial value after it has reached the threshold of 4 and associated conditions.

Other case, the process makes decisions depending on the value of the counter and the threshold variable. The process of handling different cases by returning specific values or proceeding to the next step based on the type of priority. After that, the process returns to step 5 to invoke further. Finally, checks whether a data structure (packet) is empty and does not contain priority, then terminate the algorithm.

3.3.5.1 Proposed Model (ML-MLQ)

Upon successfully loading the dataset and completing the training process, the K-nearest neighbors (KNN) model will be utilized for prediction purposes. The model will assign a priority value ranging from 0 to 4. Subsequently, the obtained priority value will be forwarded to the multilevel queue to facilitate the execution of subsequent operations. The suggested model, as described in Algorithm (3.3), is based on integrating two algorithms, namely Machine Learning-Multilevel Queue (KNN-MLQ), the steps are as follows

Steps 1–2 include the sequential data loading, preprocessing procedure for machine learning, model training using a K-NN classifier, and the following prediction process for determining priorities and calculating priority values using predicted class labels.

Steps 3–6 encompass the priority queue, which is initialized with a range of 0 to 4, and counters are set up to keep track of the packets in the queues. Push and pop operations are specified to manage the packets in the queues.

Step 7: The loop initiates and invokes the push and pop procedures throughout each iteration, followed by incrementing the counter. The code verifies whether the counter has reached thresholds (threshold1, threshold2, threshold3, threshold4). If the queues corresponding to these thresholds are not empty, the function will return the relevant value (1, 2, 3, or 4). If the counter hits a threshold of 4 and all queues are empty, the counter is reset to 0. Subsequently, the code examines the priority queues to determine if any are empty; if so, it retrieves the priority value. If no elements exist in any of the queues, the function will return None

Algorithm 3.3: Proposed K-NN with Multi-level Queue (MLQ) Model

```
Input:
DS: load dataset(Packets, Bytes, Packets A to B, Bytes A to B, Packets B to A, Bytes B to A)
Ts: Test size
Vs: Validation size
K: Number of neighbours
W: Represent the weight of neighbours
A: Type of algorithm used to compute the nearest neighbours
M: Used to compute a distance metric
Initialize Priority queue (0, 1, 2, 3, and 4)
Counter = 0
threshold1 = 8 ,threshold2 = 9, threshold3 = 10, threshold4 = 11
Push()
Output:
Pop()
Begin
1. Load and split the dataset
   X, y, X_train, X_test, y_train, y_test = load_and_split_dataset(Ts, Vs)
2. Train and predict with the K-NN model
   y_pred = train_and_predict_KNN(X_train, y_train, X_test, K, W, A, M)
   Priority  = y_pred
3. queues = {0: [], 1: [], 2: [], 3: [], 4: []}
4. Setting a counter and thresholds
5. Call the push()
  packet = get_next_packet()
  priority = get_packet_priority(packet)
  push(packet, priority)
6. Call the pop()
  popped_packet = pop(queues)
  process_popped_packet(popped_packet)
7. while True:
      Call the push()
      Call the pop()
       counter += 1
      Check the status of the thresholds
      if counter == threshold1 and queues[1]:
         return 1
      elif counter == threshold2 and queues[2]:
         return 2
      elif counter == threshold3 and queues[3]:
         return 3
      elif counter == threshold4 and queues[4]:
         return 4
      elif counter == threshold4 and not any(queues.values()):
         counter = 0
      Check priority queues
     for priority in range(5):
         if queues[priority]:
            return priority
      Check if packet is empty
      if not any(queues.values()):
         return None
```

3.3.6 Stage 6 - Load Balancing Scheduling (LBS)

The dissertation develops and proposes two algorithms based on the principles of dynamic load-balancing scheduling: the first is Least Resource Load (LRL), and the second is Hybrid Least Connection and Least Resource Load (LCLRL). The LRL algorithm calculates the least use of CPU and memory usage. On the other hand, the LCLRL algorithm combines the least connection and the least CPU and memory usage to achieve load balancing and compare it with Random (R), Round Robin (RR), and Weight Round Robin (WRR), Least Connection (LC).

3.3.6.1 Proposed of Least Resource Load (LRL) Algorithm

The LRL (Least Resource Load) method has been developed to efficiently allocate incoming network traffic across servers to minimize the utilization of CPU and memory resources. The technique presented in this study aims to enhance resource consumption to achieve optimal and balanced server performance. The LRL algorithm model consistently monitors essential resource consumption parameters for every server. Typically, these metrics include the measurement of CPU use and memory distribution.

The LRL method evaluates the existing resource load of each server as it receives new requests or tasks. The load factor computation incorporates measurements of both CPU and memory consumption. The load factor is determined by calculating a weighted CPU and memory usage sum. The server selection to handle incoming requests or tasks is based on the lowest computed load factor, which is determined by considering the combined CPU and memory use. The mathematical expression denoted as equation (3.1) is utilized to compute the proposed algorithm.

$$\mathbf{LRL = min\_i((60\%) * CPU\_i + (40\%) *Memory\_i)} \qquad \mathbf{(3.1)}$$

Let LRL represent the index of the selected server with the lowest percentage of minimum usage for CPU and Memory. Let R denote the total number of incoming requests. The index i represents the current request, ranging from 1 to R. CPU_i represents the current CPU usage on the i-th server, while Memory_i represents the current memory usage on the i-th server. Additionally, min_i denotes the server(s) index with the minimum value among all servers regarding CPU and Memory usage counts. The values of 60% and 40% are weighting factors that determine the relative importance of CPU and memory utilization in the load calculation.

Algorithm 3.4: Proposed of Least Resource Load (LRL) Algorithm

```
Input:
IP Address: IP address of SDN Controller
IP Servers: List of IP Servers with associated CPU and Memory utilization
Output:
IPlb: IP Load balancer based on the least resource load (CPU and Memory)
Begin
load_balancer_algorithm(IP_Servers)
    while True:
        1. ARP: Use ARP to check server availability
        2. TLS: Track live servers for MAC addresses and ports
        3. Record flow IP addresses, ports, and servers in memory
        4. Timeout-based flow and probe expiration
        5. Monitor CPU and Memory Utilization:
          a. current_CPU, current_Memory = get_server_utilization(server)
          b. server['CPU_i'] = current_CPU
             server['Memory_i'] = current_Memory
          c. Update CPU_i and Memory_i in the server's information
        6. LRL: Direct traffic to the server with the least resource load (minimum of CPU and Memory)
          a. For each server i in IP Servers, calculate
          LRL_i = 0.6 * server['CPU_i'] + 0.4 * server['Memory_i']
          b. index_SS = min(range(len(IP_Servers)), key=lambda i: IP_Servers[i]['LRL_i'])
          c. Direct incoming traffic to the server at index SS
        7. Install switch flow table entries to redirect traffic to the designated server
        8. Track each server's total connections and resource load
        9. if receive_request():
               continue
           else:
               break
```

Algorithm 3.4 requires the input of a configuration network, which consists of the IP address of the SDN Controller and a list of IP Servers. The algorithm's output identifies the server with the lowest LRL (Least Resource Load) as the selected server (IPlb).

- Step 1 involves the responsibility of verifying server availability through the utilization of the Address Resolution Protocol (ARP). It ensures that the algorithm contains knowledge about the present accessibility of servers within the network.
- Step 2 involves monitoring MAC addresses and ports to track live servers, and this aids in the identification of servers that have an active part in network traffic.
- Step 3, the algorithm stores data relevant to the IP addresses, ports, and associated servers related to the flow in the Memory. The data above is critical in directing network traffic to the specified servers.
- Step 4 involves managing timeouts and probe expiration in the network flow scheme. The process guarantees the removal of older or inactive flows from consideration, hence helping to keep track of the current status of the network.
- Step 5 involves continually monitoring CPU and Memory use for every server listed in the IP Servers. This process aims to maintain the algorithm's access to the latest information about resource utilization, an essential consideration in the calculation of LRL, and this involves frequently updating the values of CPU_i and Memory_i and retrieving and modifying the current CPU and Memory utilization of server i.
- In Step 6, each server's LRL (Least Resource Load) is determined by employing a weighted mix of CPU and Memory use, with CPU accounting for 60% and Memory accounting for 40%. The server displaying the lowest LRL is chosen to manage incoming network traffic.

- In phase 7, the network switches are configured to reroute incoming traffic towards the server chosen in the previous step. The process guarantees that network traffic is accurately directed to the designated server.
- In Step 8, monitoring and recording the aggregate quantity of connections and the resource utilization of every server is necessary. Providing essential information for load balancing choices and this network structure facilitates network management.
- Step 9 involve verifying the availability of a request for receiving. If a request is received, the algorithm returns to Step 1 to initiate the load-balancing process's repetition. If no request is received, ending the load balancing procedure.

3.3.6.2 Proposed Hybrid Algorithm: Least Resource Load and Least Connection (LCLRL) Algorithm

The LCLRL algorithm is a load balancing method that includes resource usage such as CPU and memory and the number of active connections to determine the optimal server for managing incoming requests. The primary objective of this algorithm is to achieve balanced traffic distribution, considering individual servers' capacity and responsiveness.

The LCLRL algorithm uses the methodology of the LC algorithm to monitor the number of active connections on each server and select the server with the fewest connections. At the same time, the LRL method calculates a load factor for each server by considering the computed resource load (CPU and memory). Calculating a weighted total of both factors will yield the desired result. Equation (3.2) represents the mathematical expression to compute the proposed algorithm.

$$\mathbf{LCLRL = min\_i((40\%) * CPU\_i + (40\%) * Memory\_i + (20\%) * LC\_i)} \quad \textbf{(3.2)}$$

Let LCLRL represent the index of the selected server with the lowest percentage of minimum usage for CPU and Memory. Let R denote the total number of incoming requests. The index i represents the current request, ranging from 1 to R. CPU_i represents the current CPU usage on the i-th server. In contrast, Memory_i represents the current memory usage on the i-th server, while LC_i represents the current number of connections on the i-th server. Additionally, min_i denotes the server index with the minimum value among all servers regarding CPU and Memory usage counts and the fewest connections. The values of 40%, 40%, and 20% are weighting factors determining the relative importance of CPU and memory and the number of connections utilized in the load calculation.

Algorithm 3.5: Proposed Combine Algorithm: Least Connection and Least Resource Load (LCLRL)

```
Input:
IP Address: IP address of SDN Controller
IP Servers: List of IP Servers with associated CPU and Memory utilization
Output:
IPlb: IP Load balancer based on the least resource load (CPU and Memory) and Least Connection
Begin
load_balancer_algorithm(IP_Servers)
    while True:
            1. ARP: Use ARP to check server availability
            2. TLS: Track live servers for MAC addresses and ports
            3. Record flow IP addresses, ports, and servers in memory
            4. Timeout-based flow and probe expiration
            5. Monitor and Retrieve CPU and Memory Utilization and Current Number of Connections
              a. current_CPU, current_Memory = get_server_utilization(server)
              b. current_LC = get_server_connections(server)
              c. server['CPU_i'] = current_CPU
                 server['Memory_i'] = current_Memory
                 server['LC_i'] = current_LC
              d. Update CPU_i and Memory_i and  LC_i in the server's information
            6. LRL: Direct traffic to the server with the least resource load (minimum of CPU and Memory)
              a. For each server i in IP Servers, calculate
                 Server['LRLLC_i'] = 0.4 * server['CPU_i'] + 0.4 * server['Memory_i'] + 0.2 * server['LC_i']
              b. index_SS = min(range(len(IP_Servers)), key=lambda i: IP_Servers[i]['LRLLC_i'])
              c.  install_switch_flow_entries(IP_Servers[index_SS])
            7. Install switch flow table entries to redirect traffic to the designated server
            8. Track each server's total connections and resource load
            9. if receive_request():
                    continue
                else:
                    break
```

Algorithm 3.5 requires the input of a configuration network, which consists of the IP address of the SDN Controller and a list of IP Servers. The algorithm's output identifies the server with the lowest combined load for LCLRL from calculate the Least Resource Load (LRL) considering CPU, Memory, and the Least Connection (LC) factor as the selected server (IPlb).

- Step 1 involves the responsibility of verifying server availability through the utilization of the Address Resolution Protocol (ARP). It ensures that the algorithm contains knowledge about the present accessibility of servers within the network.
- Step 2 involves monitoring MAC addresses and ports to track live servers, and this aids in the identification of servers that have an active part in network traffic.
- Step 3, the algorithm stores data relevant to the IP addresses, ports, and associated servers related to the flow in the Memory. The data above is critical in directing network traffic to the specified servers.
- Step 4 involves managing timeouts and probe expiration in the network flow scheme. The process guarantees the removal of older or inactive flows from consideration, hence helping to keep track of the current status of the network.
- Step 5 involves continually monitoring CPU and Memory and calculating the least connection use for every server listed in the IP Servers. This process aims to maintain the algorithm's access to the latest information about resource utilization, an essential consideration in the calculation of LCLRL, and this involves frequently updating the values of CPU_i and Memory_i and LC_i, retrieving and modifying the current CPU and Memory utilization and a current number of connections of server i.
- In Step 6, LCLRL is determined by employing a weighted mix of CPU and Memory use and the least number of active connections, with CPU accounting for 40% and Memory accounting for 40% and the least connection accounting

for 20%. The server displaying the lowest LCLRL is chosen to manage incoming network traffic.

- In phase 7, the network switches are configured to reroute incoming traffic towards the server chosen in the previous step. The process guarantees that network traffic is accurately directed to the designated server.
- In Step 8, monitoring and recording the aggregate quantity of connections and the resource utilization of every server is necessary. Providing essential information for load balancing choices and this network structure facilitates network management.
- Step 9 involve verifying the availability of a request for receiving. If a request is received, the algorithm returns to Step 1 to initiate the load-balancing process's repetition. If no request is received, ending the load balancing procedure.

3.3.7 Stage 7 - Proposed Model (ML-MLQLRL)

The proposed model algorithm integrates three techniques: Machine Learning with Multilevel Queue and Load Balancing Scheduling (ML-MLQLBS). This integration aims to develop an intelligent model called KNN-MLQLRL, which facilitates intelligent dynamic load balancing. Algorithm 3.6 represents the steps of the proposed model.

Steps of Algorithm 3.6: Proposed Model (KNN-MLQLRL)

```
Input:
IP Address: IP address of SDN Controller
IP Servers: List of IP Servers
KNN.pkl: Training Model
Output:
KNN-MLQLRL: An Intelligent Dynamic Load Balancing Scheduling
Begin
while True:
        1. ARP: Using ARP to check server availability
        2. TLS: Use of Tracking live Server for MAC addresses and ports
        3. Recording flow IP addresses, ports, and servers in memory
        4. Timeout-based flow and probe expiration
        5. Receive packets, including to URL and Features
        6. Extracted of features: Extracted features from URL
        7. Prediction: Recall training Model based on algorithm 3.2: KNN.pkl
        8. Apply the training model (KNN.pkl) to features to predict priorities
        9. Priority = y_pred (0, 1, 2, 3, or 4)
        10. Sent priority to MLQ
        11. MLQ:  Recall of MLQ model, including push and pop based on algorithm 3.3
        12. LRL: Recall algorithm 3.5 based on the least resource load algorithm
        13. Installing switch flow table entries to redirect traffic
        14 Directing traffic with features to the server based on priority and on the result of LRL
        15. Check to receive the request
        if receive_request():
            continue
        else:
            break
```

Algorithm 3.6 requires the input of a configuration network, which consists of the IP address of the SDN Controller, a list of IP Servers, and the recall of the training model. The algorithm's output identifies the server selection based on an intelligent dynamic load balancing schedule.

The algorithm begins by initializing and entering a repetitive loop, which continues until the request is received or not.

- Steps 1-6 represent verifying server availability through the utilization of the Address Resolution Protocol (ARP), monitoring MAC addresses and ports to track live servers to ensure servers are an active part of network traffic, and storing data relevant to the IP addresses, ports, and associated servers related to the flow in the Memory. The data above is critical in directing network traffic to the specified servers and also involves managing timeouts and probe expiration in the network flow scheme. The process guarantees the removal of older or inactive flows, after receiving packets that include the URL and features, Note that the URL is represent the IP SDN controller and, then extracting the features from the packet to use in the next step in the training model.
- Steps 7–10 use machine learning classification (algorithm 3.2), which includes using the extracted features to make predictions using the K-Nearest Neighbors (KNN) that have already been trained. The training model denoted as KNN.pkl, will be utilized to make predictions about priority based on the extracted features. The output is saved in the variable y_pred, representing discrete values ranging from 0 to 4. Finally, the predicted priority value (y_pred) should be transmitted to a Multi-Level Queue (MLQ) for the next step.
- Step 11 Manage the Multi-Level Queue (MLQ) using queue operations such as push to store the priorities according to the type and level of priorities and pop to send them to LRL, based on algorithm 3.3
- In Step 12, the Least Resource Load (LRL) algorithm, derived from algorithm 3.5, determines the server with the least resource load. This determination is made by calculating the lowest CPU and memory consumption.

- In steps 13-15 the task at hand involves the configuration of a switch flow table to route incoming traffic to the server effectively. This routing process will be based on two key factors: the prediction priority and the outcome of the Least Resource Load (LRL) algorithm. Ultimately, it is necessary to ascertain whether or not a new request has been received. If a request is received, it is necessary to revert back to Step 1 to initiate the repetition of the load balancing procedure. If no request is received, terminate the loop.

Chapter Four

Results and Discussion

4.1 Overview

The proposed model is explained theoretically in the previous chapter, and the results are presented and discussed in this chapter. Two types of datasets have been applied as employment case studies to determine the behavior of the proposed model. Furthermore, the experimental stages of the proposed model are described and shown in this chapter. Before starting to analyze the results of the proposed model, it is crucial to present its general characteristics:

1. Generation of traffic based on two types of files to create datasets 1 and 2.
2. It deals with real files (Facebook files) to create datasets in the SDN network.
3. It depends on machine learning principles.
4. It builds rules for the queue model.
5. It processes the problem of packet starvation that occurs in the queue model
6. It deals with the resources of servers (CPU and Memory)
7. It constructs intelligent dynamic load balancing.

4.2 System Requirement and Network Topology

The model requirements related to the efficient and optimal functioning of the software and SDN environments are explained in Chapter 3, Section 3.4 of this dissertation. The requirements above involve different elements, including hardware specifications, the operating system and simulation selection, and the suggested network topology. In addition, this section will explain the various requirements of a programming language, including the integrated development environment (IDE). Table 4.1 provides a comprehensive overview of the model's needs and network structure.

Table 4.1: Overview of the Model Requirement and Network Structure

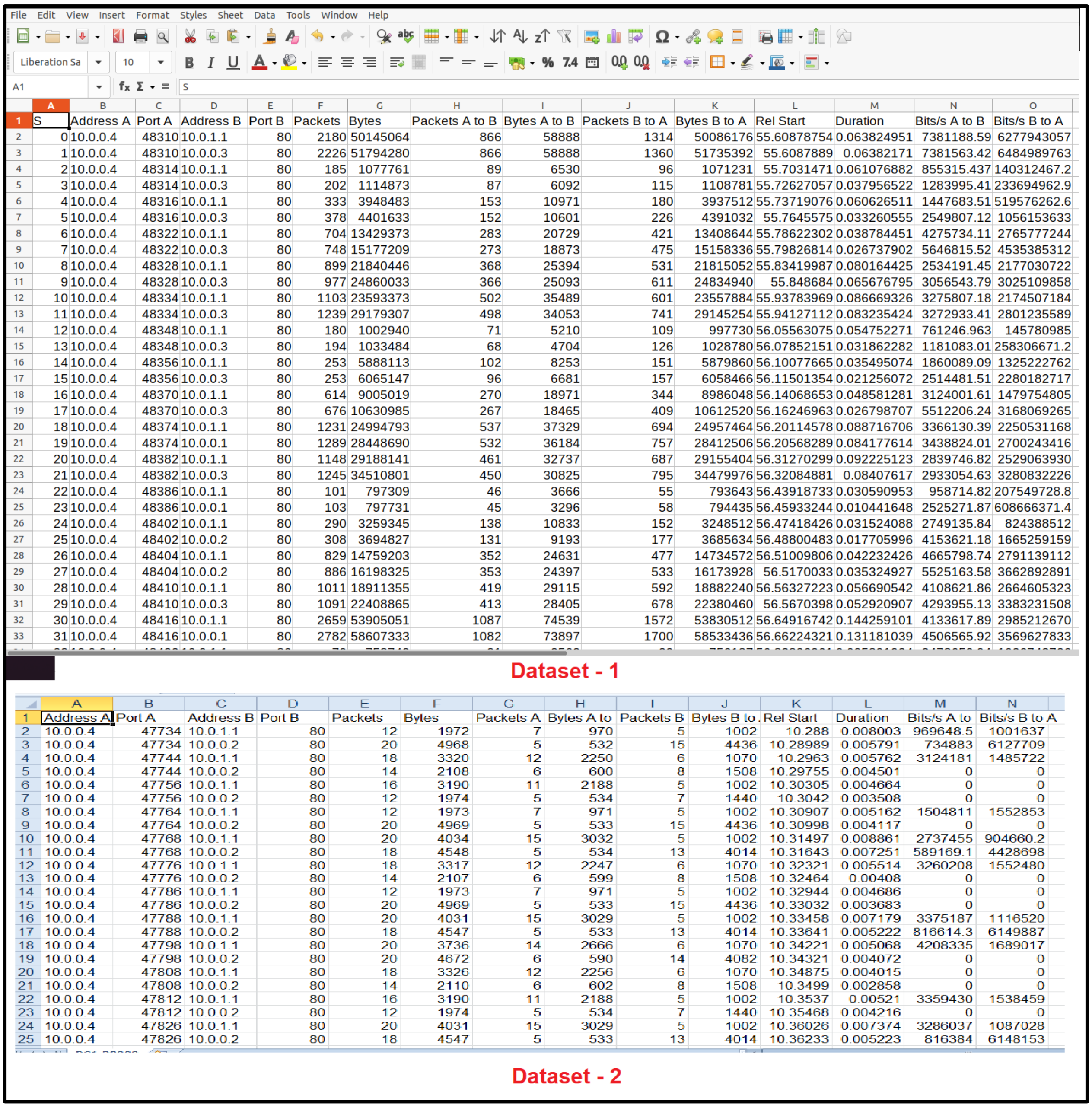
Requirements	Details
Hardware	Using two hardware 1. Processor: Intel Core I7-3625QM 2.2GHz *8, RAM: 8 GB, Storage: 512GB SSD, Graphics: AMD/Intel 4000 2. Processor: Intel Core I3-4030U 1.90GHz, RAM: 6GB, Storage: 512GB SSD, Graphics: NVIDIA/Intel
Operation System (OS)	Using two OS 1. Linux Ubuntu 20.04.4 LTS/64-bit: Install on hardware1 2. Windows 10 Pro 64-bit: Install on hardware 2
Software	Wireshark is a network traffic analyzer that is installed, runs on Linux, and works with Mininet to capture, analyze, and inspect network traffic.
Emulator	Mininet / running in Linux
Programming Language	Python 3.9
IDE	Anaconda / Spyder 5.1.5 (Python 3.9) running on Windows
Network Topology	The proposed flat topology includes: -Single SDN controller, specifically the POX-Controller, written in Python. -Single switch, the type is an open virtual switch(OVS) -Open-Flow (OF) protocol to establish connections between the controller and switch -Three hosts (H1, H2, and H3) -Three servers (S1, S2, and S3)

4.3 Description of the SDN Network Datasets

This section shows how to create the datasets used in the current study. As mentioned in Chapter 2, Section 2.9, and Table 2.3, two types of files will be used to create two datasets. The primary objective of this dissertation is to produce an extensive dataset through the execution of packet generation, capture, analysis, and collection. In the first category (type 1), files are created manually and are used to generate traffic, including five files. In contrast, the second category (type 2) encompasses real files obtained from the University of California and collected from the Facebook website, including 16 files. The files mentioned above are utilized to generate traffic and then capture and collect these files to create two datasets and store them as Excel files. As mentioned in the previous chapter, Section 3.3.1, the data generation stage includes four steps: request generation, traffic capturing, data collection, and dataset creation. Also, Wireshark will be used as a network traffic analyzer. Finally, after collecting the data, it is stored as Excel files to create a dataset. Table 4.2 represents an overview of all the details of the generation of files and datasets. Additionally, Figure 4.1 shows datasets collected using Wireshark and stored as an Excel file

Table 4.2 Summary of File and Dataset Generation Details

Requests	Source	Details	No. Files	Dataset Type	Dimension
Type 1	Flow Generation by Creating Files Manually	Creating five files of different sizes (1 Mbyte, 10 Mbyte, 25 Mbyte, 50 Mbyte, and 75 Mbyte)	5	Dataset 1	6 Features * 4184 rows
Type 2	Real Files (Facebook) - University of California, San Diego	18% of the data was less than 0.1 Kbyte (3 files), 19%–38% was between 0.1 and 10 Kbyte (3 files), 39%–89% was 1 Mbyte (8 files), and 90%–100% was less than 10 Mbyte (2 files).	16	Dataset 2	6 Features * 30,000 rows



S	Address A	Port A	Address B	Port B	Packets	Bytes	Packets A to B	Bytes A to B	Packets B to A	Bytes B to A	Rel Start	Duration	Bits/s A to B	Bits/s B to A
0	10.0.0.4	48310	10.0.1.1	80	2180	50145064	866	58888	1314	50086176	55.60878754	0.063824951	7381188.59	6277943057
1	10.0.0.4	48310	10.0.0.3	80	2226	51794280	866	58888	1360	51735392	55.6087889	0.06382171	7381563.42	6484989763
2	10.0.0.4	48314	10.0.1.1	80	185	1077761	89	6530	96	1071231	55.7031471	0.061076882	855315.437	140312467.2
3	10.0.0.4	48314	10.0.0.3	80	202	1114873	87	6092	115	1108781	55.72627057	0.037956522	1283995.41	233694962.9
4	10.0.0.4	48316	10.0.1.1	80	333	3948483	153	10971	180	3937512	55.73719076	0.060626511	1447683.51	519576262.6
5	10.0.0.4	48316	10.0.0.3	80	378	4401633	152	10601	226	4391032	55.7645575	0.033260555	2549807.12	1056153633
6	10.0.0.4	48322	10.0.1.1	80	704	13429373	283	20729	421	13408644	55.78622302	0.038784451	4275734.11	2765777244
7	10.0.0.4	48322	10.0.0.3	80	748	15177209	273	18873	475	15158336	55.79826814	0.026737902	5646815.52	4535385312
8	10.0.0.4	48328	10.0.1.1	80	899	21840446	368	25394	531	21815052	55.83419987	0.080164425	2534191.45	2177030722
9	10.0.0.4	48328	10.0.0.3	80	977	24860033	366	25093	611	24834940	55.848684	0.065676795	3056543.79	3025109858
10	10.0.0.4	48334	10.0.1.1	80	1103	23593373	502	35489	601	23557884	55.93783969	0.086669326	3275807.18	2174507184
11	10.0.0.4	48334	10.0.0.3	80	1239	29179307	498	34053	741	29145254	55.94127112	0.083235424	3272933.41	2801235589
12	10.0.0.4	48348	10.0.1.1	80	180	1002940	71	5210	109	997730	56.05563075	0.054752271	761246.963	145780985
13	10.0.0.4	48348	10.0.0.3	80	194	1033484	68	4704	126	1028780	56.07852151	0.031862282	1181083.01	258306671.2
14	10.0.0.4	48356	10.0.1.1	80	253	5888113	102	8253	151	5879860	56.10077665	0.035495074	1860089.09	1325222762
15	10.0.0.4	48356	10.0.0.3	80	253	6065147	96	6681	157	6058466	56.11501354	0.021256072	2514481.51	2280182717
16	10.0.0.4	48370	10.0.1.1	80	614	9005019	270	18971	344	8986048	56.14068653	0.048581281	3124001.61	1479754805
17	10.0.0.4	48370	10.0.0.3	80	676	10630985	267	18465	409	10612520	56.16246963	0.026798707	5512206.24	3168069265
18	10.0.0.4	48374	10.0.1.1	80	1231	24994793	537	37329	694	24957464	56.20114578	0.088716706	3366130.39	2250531168
19	10.0.0.4	48374	10.0.0.1	80	1289	28448690	532	36184	757	28412506	56.20568289	0.084177614	3438824.01	2700243416
20	10.0.0.4	48382	10.0.1.1	80	1148	29188141	461	32737	687	29155404	56.31270299	0.092225123	2839746.82	2529063930
21	10.0.0.4	48382	10.0.0.3	80	1245	34510801	450	30825	795	34479976	56.32084881	0.08407617	2933054.63	3280832226
22	10.0.0.4	48386	10.0.1.1	80	101	797309	46	3666	55	793643	56.43918733	0.030590953	958714.82	207549728.8
23	10.0.0.4	48386	10.0.0.1	80	103	797731	45	3296	58	794435	56.45933244	0.010441648	2525271.87	608666371.4
24	10.0.0.4	48402	10.0.1.1	80	290	3259345	138	10833	152	3248512	56.47418426	0.031524088	2749135.84	824388512
25	10.0.0.4	48402	10.0.0.2	80	308	3694827	131	9193	177	3685634	56.48800483	0.017705996	4153621.18	1665259159
26	10.0.0.4	48404	10.0.1.1	80	829	14759203	352	24631	477	14734572	56.51009806	0.042232426	4665798.74	2791139112
27	10.0.0.4	48404	10.0.0.2	80	886	16198325	353	24397	533	16173928	56.5170033	0.035324927	5525163.58	3662892891
28	10.0.0.4	48410	10.0.1.1	80	1011	18911355	419	29115	592	18882240	56.56327223	0.056690542	4108621.86	2664605323
29	10.0.0.4	48410	10.0.0.3	80	1091	22408865	413	28405	678	22380460	56.5670398	0.052920907	4293955.13	3383231508
30	10.0.0.4	48416	10.0.1.1	80	2659	53905051	1087	74539	1572	53830512	56.64916742	0.144259101	4133617.89	2985212670
31	10.0.0.4	48416	10.0.0.1	80	2782	58607333	1082	73897	1700	58533436	56.66224321	0.131181039	4506565.92	3569627833

Dataset - 1

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A	Bytes A to	Packets B	Bytes B to.	Rel Start	Duration	Bits/s A to	Bits/s B to A
10.0.0.4	47734	10.0.1.1	80	12	1972	7	970	5	1002	10.288	0.008003	969648.5	1001637
10.0.0.4	47734	10.0.0.2	80	20	4968	5	532	15	4436	10.28989	0.005791	734883	6127709
10.0.0.4	47744	10.0.1.1	80	18	3320	12	2250	6	1070	10.2963	0.005762	3124181	1485722
10.0.0.4	47744	10.0.0.2	80	14	2108	6	600	8	1508	10.29755	0.004501	0	0
10.0.0.4	47756	10.0.1.1	80	16	3190	11	2188	5	1002	10.30305	0.004664	0	0
10.0.0.4	47756	10.0.0.2	80	12	1974	5	534	7	1440	10.3042	0.003508	0	0
10.0.0.4	47764	10.0.1.1	80	12	1973	7	971	5	1002	10.30907	0.005162	1504811	1552853
10.0.0.4	47764	10.0.0.2	80	20	4969	5	533	15	4436	10.30998	0.004117	0	0
10.0.0.4	47768	10.0.1.1	80	20	4034	15	3032	5	1002	10.31497	0.008861	2737455	904660.2
10.0.0.4	47768	10.0.0.2	80	18	4548	5	534	13	4014	10.31643	0.007251	589169.1	4428698
10.0.0.4	47776	10.0.1.1	80	18	3317	12	2247	6	1070	10.32321	0.005514	3260208	1552480
10.0.0.4	47776	10.0.0.2	80	14	2107	6	599	8	1508	10.32464	0.00408	0	0
10.0.0.4	47786	10.0.1.1	80	12	1973	7	971	5	1002	10.32944	0.004686	0	0
10.0.0.4	47786	10.0.0.2	80	20	4969	5	533	15	4436	10.33032	0.003683	0	0
10.0.0.4	47788	10.0.1.1	80	20	4031	15	3029	5	1002	10.33458	0.007179	3375187	1116520
10.0.0.4	47788	10.0.0.2	80	18	4547	5	533	13	4014	10.33641	0.005222	816614.3	6149887
10.0.0.4	47798	10.0.1.1	80	20	3736	14	2666	6	1070	10.34221	0.005068	4208335	1689017
10.0.0.4	47798	10.0.0.2	80	20	4672	6	590	14	4082	10.34321	0.004072	0	0
10.0.0.4	47808	10.0.1.1	80	18	3326	12	2256	6	1070	10.34875	0.004015	0	0
10.0.0.4	47808	10.0.0.2	80	14	2110	6	602	8	1508	10.3499	0.002858	0	0
10.0.0.4	47812	10.0.1.1	80	16	3190	11	2188	5	1002	10.3537	0.00521	3359430	1538459
10.0.0.4	47812	10.0.0.2	80	12	1974	5	534	7	1440	10.35468	0.004216	0	0
10.0.0.4	47826	10.0.1.1	80	20	4031	15	3029	5	1002	10.36026	0.007374	3286037	1087028
10.0.0.4	47826	10.0.0.2	80	18	4547	5	533	13	4014	10.36233	0.005223	816384	6148153

Dataset - 2

Figure 4.1: Create Datasets 1 and 2 Based on Requests (Types 1 and 2)

4.4 The Results of the Preprocessing Stage

This stage involves transforming the raw data already collected into carefully organized and structured data. Chapter 3, Section 3.3.2, outlines that data conversion necessitates implementing two distinct stages, each employing a designated methodology. During the initial stage, the degree of correlation between columns (features). This technique transforms all values in the correlation matrix to a range between 0 and 1 and subsequently performs feature selection by prioritizing values that are near 1. Figure 4.2 illustrates the values for the dataset's features that the correlation matrix determined, ranging between 0 and 1. In this particular case, it can be determined that a correlation value closer to 1 signifies a stronger correlation. Consequently, for datasets 1 and 2, six features will be chosen. These features are Packets, Bytes, Packets A to B, Bytes A to B, Packets B to A, Bytes B to A.

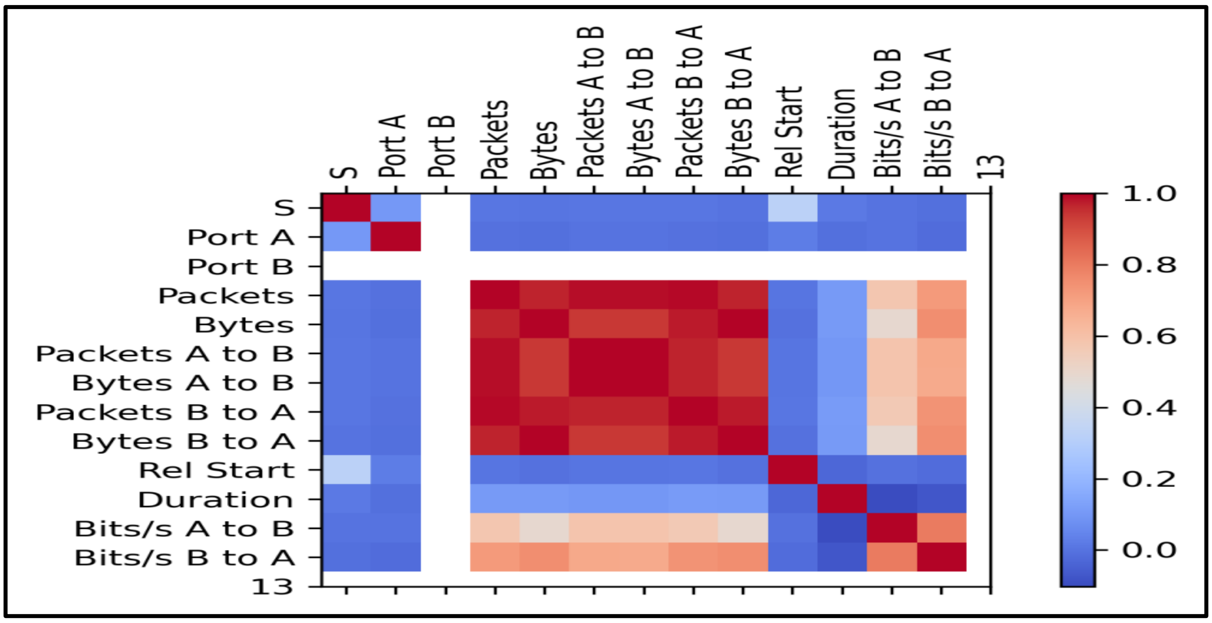


Figure 4.2: Feature Selection Scores Across Datasets

In the second phase, the dissertation proposes utilizing three primary techniques: the MinMaxScaler, the StandardScaler, and the RobustScaler. These selections are mention on Chapter 3, Section 3.3.2.3. The selection process eventually determines the most suitable technique by evaluating the standard deviation, a widely employed metric for quantifying the dispersion or diversity of data points within an individual feature. The objective is to select the most suitable dataset by considering the results with the lowest average standard deviation across all features.

The results of computing the average standard deviation for every feature in both datasets using the three strategies (MinMaxScaler, StandarScaler, and RobustScaler) with datasets 1 and 2 are presented in Table 4.3. Figure 4.3 briefly overviews the average standard deviation of outcomes obtained by implementing three strategies. The MinMaxScaler technique achieves the best result by calculating the lowest standard deviation for both datasets. In dataset 1, the standard deviation is 0.17889, whereas for dataset 2, it is 0.17968.

Table 4.3: Summary of Average Standard Deviation Results Using Three Strategies

Dataset 1	MinMax Scaler	Standard Scaler	Robust Scaler	Dataset 2	MinMax Scaler	Standard Scaler	Robust Scaler
Feature 1	0.18043	1.00012	0.67854	Feature 1	0.20912	1.000016	0.52281
Feature 2	0.19233	1.00012	0.67637	Feature 2	0.11076	1.000016	0.53727
Feature 3	0.16643	1.00012	0.69022	Feature 3	0.20200	1.000017	0.53868
Feature 4	0.16650	1.00012	0.68980	Feature 4	0.14458	1.000016	0.57341
Feature 5	0.17531	1.00012	0.66872	Feature 5	0.28661	1.000018	0.46574
Feature 6	0.19235	1.00012	0.67651	Feature 6	0.12502	1.000017	0.52340
Average	0.17889	1.00012	0.68003	Average	0.17968	1.000016	0.52688

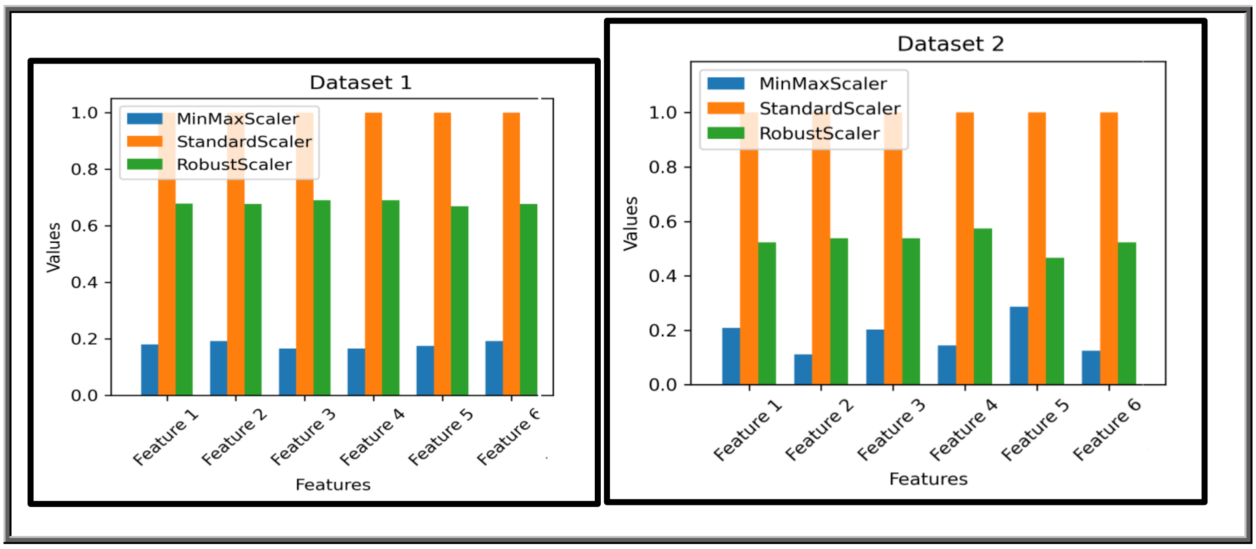


Figure 4.3: Dataset Standard Deviation Analysis Using Three Scaling Strategies

4.5 Evaluating the Clustering Algorithms

The present study considers the clustering category using three fundamental classifications. These classifications involve utilizing five algorithms designed to analyze unlabeled data to identify patterns, structures, or relationships within the data. Subsequently, labels are assigned to datasets 1 and 2. The categories mentioned above, algorithms, and equations utilized by the algorithm for clustering to create class labeling are as follows: The first category of algorithms is known as partitioning algorithms, namely the K-means algorithm.

The second category, density-based algorithms, includes DBSCAN and OPTICS. The third category, distribution-based algorithms, encompasses GMM. The proposed algorithm is a hybrid approach that combines the OPTICS and GMM techniques.

Determining the most suitable algorithm for assigning class labels depends on various variables. Several factors need to be considered while creating clusters, such as achieving an equal distribution of data across clusters, reducing the number of outliers or noise, and determining the appropriate number of clusters. The general shape of the data distribution is an essential consideration in selecting an algorithm if the shape is spherical, irregular, or random. Scatter plots and other visualizations identify it. All of these factors together contribute to the determination of the most suitable algorithm.

4.5.1 The Results of Datasets 1 and 2

Previously, the optimal approach based on calculating the average standard deviation of the lowest value choosing the best technique is MinMaxScaler in the previous section, specifically Section 4.4. The dissertation also suggest an alternative approach for determining the optimal strategy, which is the silhouette score, in addition to depending on the computation of the mean standard deviation. The previous approach focused on quantifying the dispersion or diversity of data points within a single feature without considering the relationships and similarities between data points across clusters.

In contrast, the silhouette score and the utilization of equations 2.6 and 2.7 aim to measure the similarity of each data point in a dataset to its cluster (cohesion) compared to other clusters (separation). By calculating this score for three datasets, it is possible to compare the clustering performance of different datasets and identify which produces more internally coherent and wellseparated clusters. Table 4.4 presents the silhouette scores of the dataset 1and 2 obtained using various methodologies and algorithms.

Table 4.4: Silhouette Score for Different Technique and Algorithms in Dataset 1

Dataset 1			
Silhouette Score / Algorithms	**MinMax Scaler**	**Standard Scaler**	**Robust Scaler**
K-Means	0.38	0.38	0.37
DBCSAN	0.0017	0.0002	-0.0165
Optics	-0.4883	-0.5214	-0.4949
GMM	0.191	0.1751	0.1696
Proposed (Optics + GMM)	0.1408	0.1181	0.0297
Dataset 2			
Silhouette Score / Algorithms	**MinMax Scaler**	**Standard Scaler**	**Robust Scaler**
K-Means	0.53	0.49	0.48
DBCSAN	-0.1315	-0.1939	-0.0323
Optics	0.03	-0.0729	-0.1432
GMM	0.4556	0.2679	0.373
Optics + GMM	0.1276	0.076	0.0259

Determining the most suitable algorithm for assigning class labels depends on various factors. When designing clusters, it is essential to consider multiple elements, one of which is the general shape of the data distribution. The distribution's spherical, irregular, or random shape is crucial in determining the most suitable data modeling approach. Figure 4.4 presents a graphical depiction of the overall pattern of data point distribution in datasets 1 and 2. This graphic provides an initial viewpoint that helps identify the algorithms most suited for accurately modeling the observed data shape.

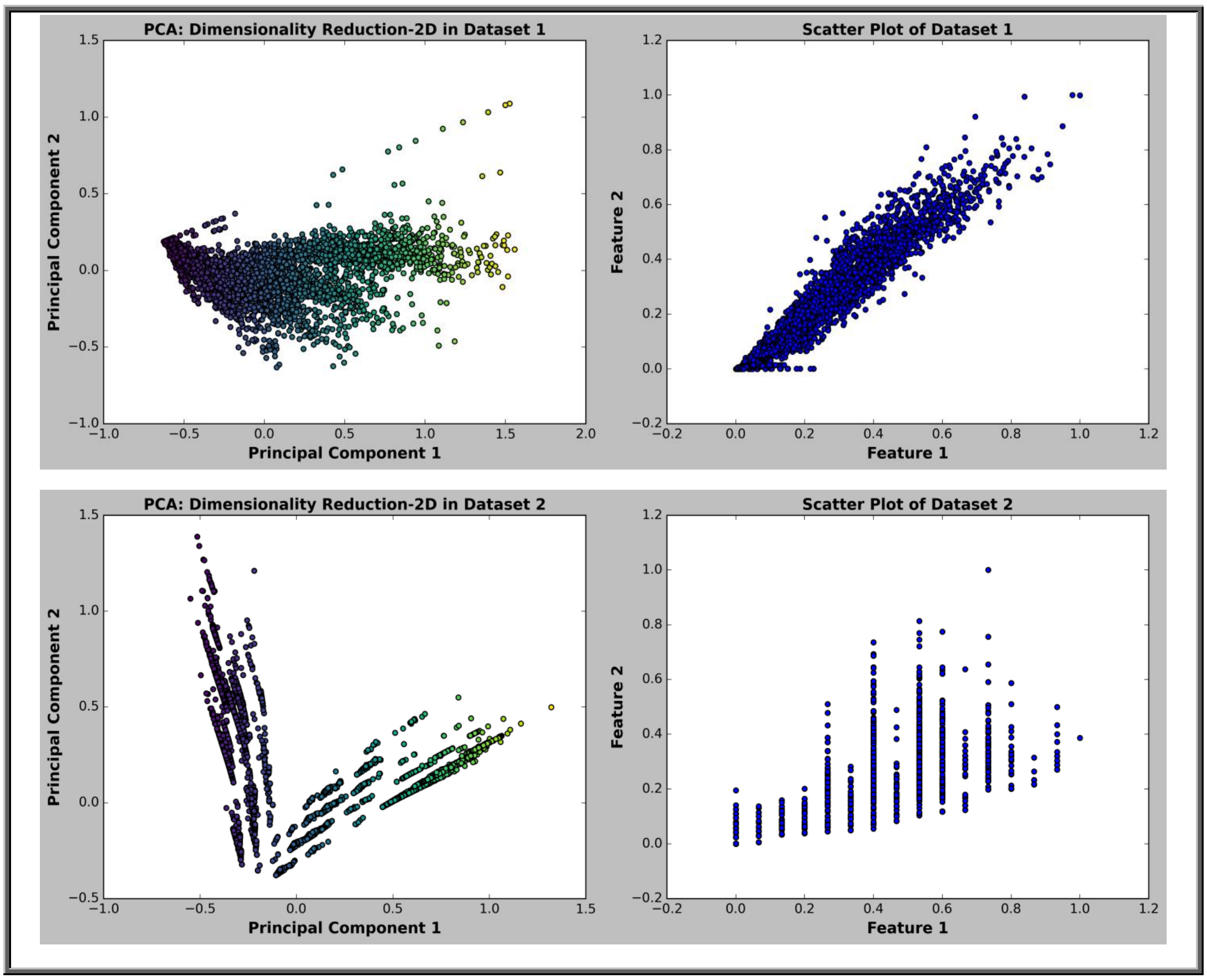


Figure 4.4: Visualization of Data Point Distribution in Datasets 1 and 2

Achieving a balanced data distribution among clusters is a crucial component that contributes to reducing outliers or noise. Determining the number of outliers or noise values is of significant importance when implementing networks since the concept of load balancing to enhance the distribution of workloads among servers within a network represents a valid and crucial use environment.

The optimization of resource utilization, improvement of Quality of Service (QoS), and efficient operation of network services are significantly influenced by load balancing. In this particular scenario, identifying and managing noise (anomalies or outliers) is of significant importance in resource allocation, quality of service improvement, and resource optimization. In the context of resource allocation, noise points can be defined as unusual or unanticipated increases in network traffic that may disturb the resource allocation process. Identifying and managing noise points can contribute to the effective allocation of resources, reducing the risks associated with server overload or poor utilization.

In the Quality of Service (QoS) enhancement, load balancing aims to enhance QoS by equally dividing the workload among multiple servers. A high number of noise points has the potential to introduce unpredictable fluctuations in workload, which may result in a decrease in Quality of Service (QoS). Identifying and managing these differences can result in enhanced consistency and dependability in Quality of Service (QoS). By effectively recognizing and dealing with areas of inefficiency, it is possible to enhance resource efficiency. Servers can achieve higher levels of efficiency when they are not required to handle unexpected or disruptive traffic patterns.

To efficiently achieve the load balancing objectives, consider using clustering algorithms that can handle noisy points. Algorithms such as DBSCAN, OPTICS, and the proposed Optics + GMM method are well-suited for this objective and can identify clusters of regular traffic and distinguish noise points. The K-Means and GMM are unlike other algorithms, which cannot identify and effectively manage noise points. It is important to note that this algorithm displayed higher performance in data distribution and achieved a higher silhouette score than other algorithms. However, when dealing with network load distribution and aiming to achieve factors such as resource allocation, QoS enhancement, and resource efficiency, it becomes imperative to utilize algorithms capable of effectively detecting and identifying noise.

Tables 4.5 and 4.6 present a comprehensive overview of the distribution of values across clusters for each priority of the five methods. This distribution depends on determining the number of clusters, which in this case is five.

Table 4.5: Cluster Distribution by Priority: Five Methods (Dataset 1)

Using the K-Means Algorithm: 5 Clusters								
Priority	0	1	2	3	4	Noise (-1)	Total	Hyper parameter
Number of Data Point	926	342	1123	1052	740	0	4,183	n_clusters=5
Using the DBSCAN Algorithm: 5 Clusters								
Priority	0	1	2	3	4	Noise (-1)	Total	Hyper parameter
Number of Data Point	2088	30	30	16	13	2006	4,183	eps=0.05, min_samples=16
Using the Optics Algorithm: 5 Clusters								
Priority	0	1	2	3	4	Noise (-1)	Total	Hyper parameter
Number of Data Point	35	72	46	43	43	3944	4,183	min_samples=5, xi=0.03, min_cluster_size=35
Using the GMM Algorithm: 5 Clusters								
Priority	0	1	2	3	4	Noise (-1)	Total	Hyper parameter
Number of Data Point	1472	91	950	742	928	0	4,183	n_components=5
Using the Optics + GMM Algorithm: 5 Clusters								
Priority	0	1	2	3	4	Noise (-1)	Total	Hyper parameter
Number of Data Point	880	681	1325	981	316	0	4,183	min_samples=5, xi=0.03, min_cluster_size=35 n_components=5

Table 4.6: Cluster Distribution by Priority: Five Methods (Dataset 2)

Using the K-Means Algorithm: 5 Clusters								
Priority	0	1	2	3	4	Noise (-1)	Total	Hyper parameter
Number of Data Point	2838	3857	8073	9579	5652	0	30,000	n_clusters=5
Using the DBSCAN Algorithm: 5 Clusters								
Priority	0	1	2	3	4	Noise (-1)	Total	Hyper parameter
Number of Data Point	668	562	3752	801	810	23406	30,000	eps=0.01, min_samples=550
Using the Optics Algorithm: 5 Clusters								
Priority	0	1	2	3	4	Noise (-1)	Total	Hyper parameter
Number of Data Point	3524	2708	4469	2701	3783	12814	30,000	min_samples=5, xi=0.03, minclustersize=2700
Using the GMM Algorithm: 5 Clusters								
Priority	0	1	2	3	4	Noise (-1)	Total	Hyper parameter
Number of Data Point	4447	6703	7884	4275	6690	0	30,000	n_components=5
Using the Proposed Optics + GMM Algorithm: 5 Clusters								
Priority	0	1	2	3	4	Noise (-1)	Total	Hyper parameter
Number of Data Point	7038	6019	7746	4235	4961	0	30,000	min_samples=5, xi=0.03, min_cluster_size=2700 n_components=5

The distribution of data points for algorithms capable of handling noise points is shown in Figure 4.5. Algorithms include DBSCAN, Optics, and the proposed hybrid algorithm combining Optics and GMM. The algorithm proposed by the dissertation, which combines the Optics and GMM techniques, is considered the most effective approach for processing noise and distributing data points.

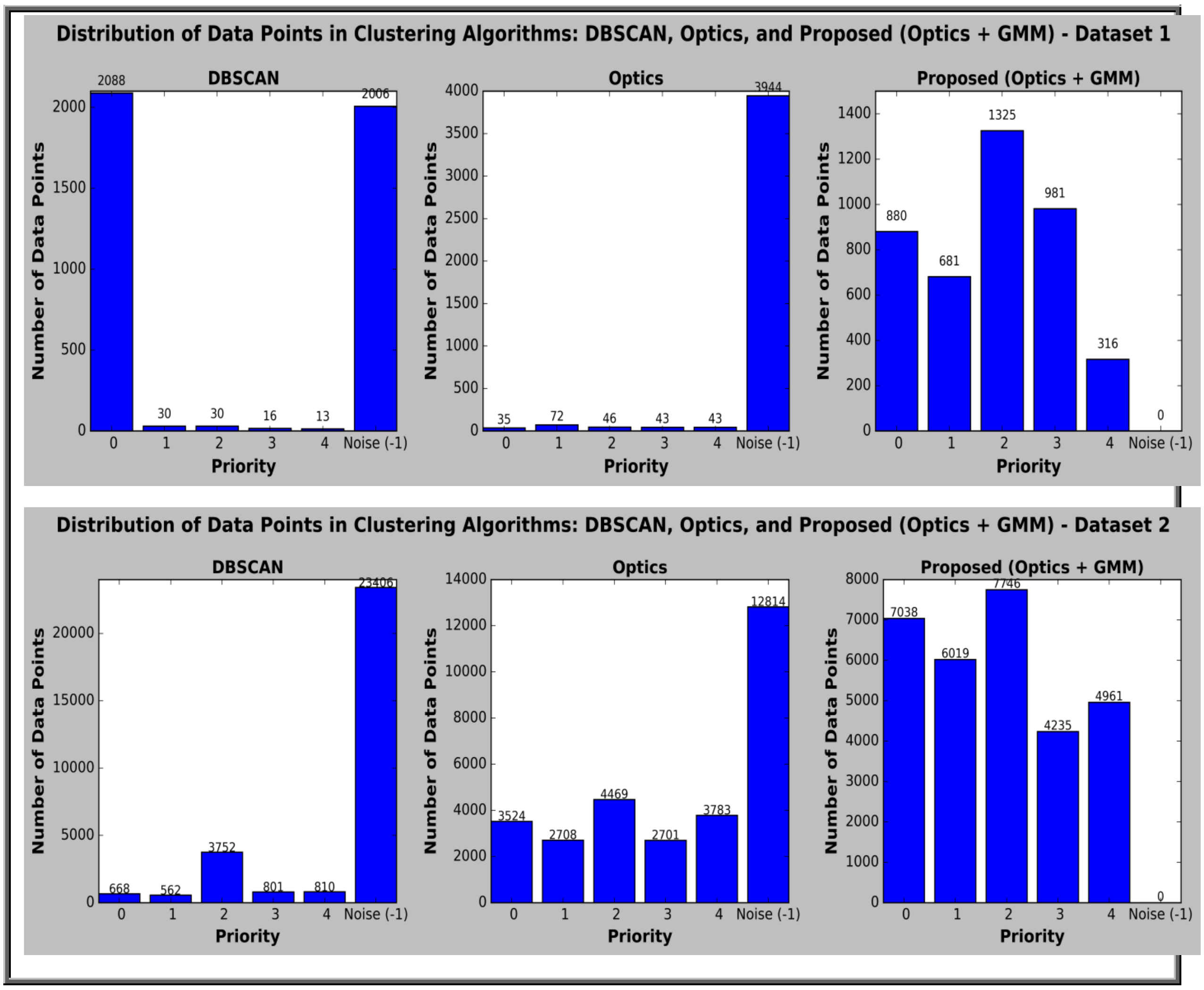


Figure 4.5: Data Point Distribution: Clustering Algorithms - Datasets 1 & 2

4.6 Evaluating the Classification Model

Supervised learning problems can be categorized into two main types: regression and classification methodologies, as Chapter 2, Section 2.5.2. This dissertation will utilize classification methodologies to deal with dataset classification and prediction. The dissertation utilizes five distinct classifiers, specifically: (1) decision tree (DT), (2) support vector machine (SVM), (3) k-nearest neighbor, (4) naïve Bayes (NB), and (5) random forest (RF). Determining the optimal algorithm for the classification model is based on its application on datasets 1 and 2 to reach high evaluation measures, including accuracy and performance metrics.

4.6.1 The Results of Datasets 1 and 2

Datasets 1 and 2 contain significant attributes, including packet size, rate, and count. These qualities play a crucial role in the classification task, with priorities assigned based on data volume and packet size. The primary aim of this dissertation is to design classification models using the datasets. This model will utilize the features of datasets to build a highly accurate predictive model.

The evaluation metrics for both Datasets 1 and 2 are presented in Tables 4.7 and 4.8, respectively. The dissertation employed a range of classification algorithms, each containing distinct hyperparameters customized to suit the specific task. Figure 4.6 presents an illustration of the accuracy of classification algorithms.

Analyzing the outcomes from the tables mentioned above, it becomes apparent that the KNN algorithm performs better than the other algorithms. In the first dataset, the KNN algorithm achieved an excellent accuracy of 94.02%. In comparison, the SVM algorithm achieved 78.39%, the DT algorithm at 70.01%, the RFC algorithm at 84.16%, and the Naive Bayes (NB) algorithm at 74.60%. In Dataset 2, the KNN algorithm displayed an excellent accuracy of 99.83%, surpassing the SVM with an accuracy of 54.65%, the DT with an accuracy of 73.63%, the FRC with an accuracy of 95.09%, and the NB with an accuracy of 72.5%. The algorithm (3.2) shows the most critical part of the K-NN technique. The K-nearest neighbors (KNN) method always did better on various performance metrics, including confusion matrices and evaluation metrics.

Table 4.7: Classification Algorithm Evaluation for Dataset 1

No.	Algorithms	Hyperparameters	Accuracy
1	Support Vector Machines (SVM)	C=0.9,kernel = 'rbf', max_iter=200, gamma='auto' test_size=0.25 random_state=43 val_size = 0.45	78.39%
2	Decision Trees (DT)	criterion= gini, max_depth=5, min_samples_split=4, min_weight_fraction_leaf=0.1, splitter='random', random_state=0 test_size=0.24 random_state=42 val_size = 0.38	70.01%
3	Random Forest Classifier (RFC)	criterion = ' entropyi', n_estimators=80, max_depth=4, min_samples_split=7, random_state=0 test_size=0.24 random_state=42 val_size = 0.50	84.16%
4	Naive Bayes (NB)	priors=None test_size=0.24 random_state=0 val_size = 0.42	74.60%
5	k-Nearest Neighbors (k-NN)	n_neighbors= 9,weights ='uniform', algorithm="auto", metric=" minkowski" test_size=0.22 random_state=27 val_size = 0.46	94.02%

Table 4.8: Classification Algorithm Evaluation for Dataset 2

No.	Algorithms	Hyperparameters	Accuracy
1	Support Vector Machines (SVM)	C=1,kernel = 'rbf', max_iter=70, gamma='auto' random_state=27 test_size=0.24 random_state=24 val_size = 0.38	54.65%
2	Decision Trees (DT)	criterion='gini', max_depth=5, min_samples_split=4, min_weight_fraction_leaf=0.1, splitter='random', random_state=42 test_size=0.23 random_state=42 val_size = 0.38	73.63%
3	Random Forest Classifier (RFC)	criterion = 'entropyi',n_estimators=100, max_depth=4, min_samples_split=7, random_state=42 test_size=0.21 random_state=42 val_size = 0.38	95.09%
4	Naive Bayes (NB)	priors=None, test_size=0.24 random_state=42, val_size = 0.42	72.5%
5	k-Nearest Neighbors (k-NN)	n_neighbors= 9, weights ='uniform', algorithm="auto", metric=" manhattani" test_size=0.25 random_state=42, val_size = 0.35	99.83%

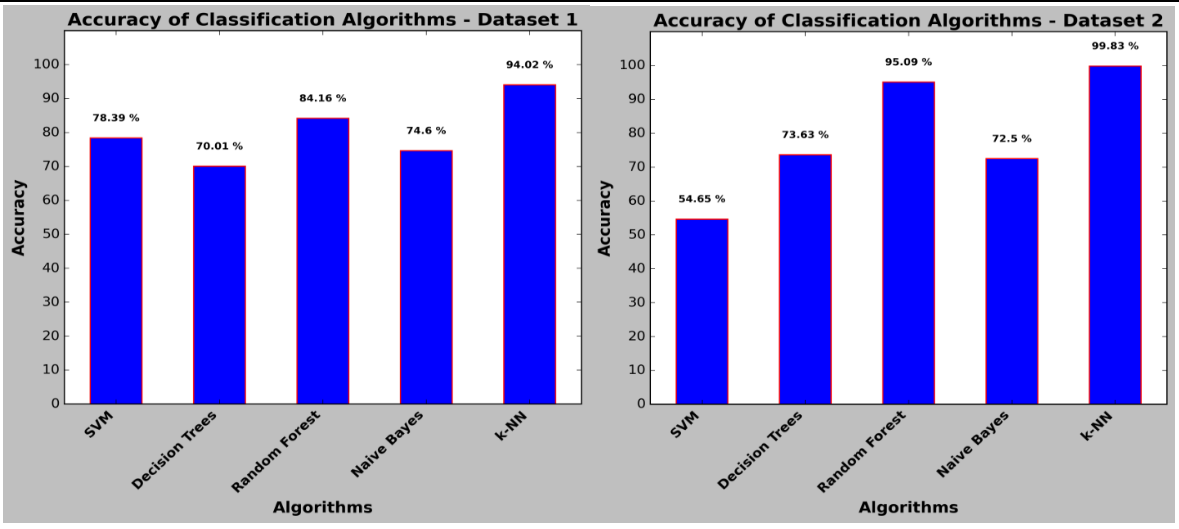


Figure 4.6: Accuracy Comparison of Classification Algorithms in Datasets 1 and 2

4.6.2 The Results of Performance Metrics

The K-nearest neighbors (KNN) technique displays excellent results compared to alternative classification algorithms. It constantly provides excellent results across many performance indicators, such as confusion matrices and evaluation criteria. Tables 4.9 and 4.10 present a complete summary of the Performance Metrics for Classification Algorithms in datasets 1 and 2, respectively. Furthermore, the graphic representations in Figures 4.7 and 4.8 provide a comparative analysis of the confusion matrices and evaluation metrics for classification algorithms in Datasets 1 and 2. These figures demonstrate the better overall performance of the K-Nearest Neighbors (KNN) algorithm in both datasets.

Table 4.9: Classification Algorithm Performance Metrics for Dataset 1

Algorithms	Support Vector Machines	Decision Trees (DT)	Random Forest Classifier
Confusion Matrix	[8 93 2 0 100] [0 11 0 155 0] [0 0 231 13 6] [0 330 0 14 0] [18 2 15 44 4]	[0 30 156 0 0] [0 11 0 138 0] [0 2 205 13 0] 0 244 39 17 0] [0 9 37 20 0]	[0 19 12 0 173] [0 3 0 164 0] [0 0 207 13 17] [2 293 0 5 23] [22 5 17 21 8]
Evaluation Metrics	Precision: 76.63% Recall: 78.39% F1 score: 75.17% Accuracy: 78.39 %	Precision: 68.15% Recall: 70.01% F1 score: 67.69% Accuracy: 70.01%	Precision: 84.12% Recall: 84.16% F1 score: 82.09% Accuracy: 84.16%
Algorithms	Naive Bayes (NB)	k-Nearest Neighbors (k-NN)	
Confusion Matrix	[0 53 25 0 129] [0 1 0 157 0] [0 2 180 13 41] [4 272 0 13 37] [12 5 23 26 11]	[1 5 0 0 182] [0 1 2 140 0] [0 0 202 1 7] [3 302 0 2 5] [7 1 7 13 40]	
Evaluation Metrics	Precision: 76.63% Recall: 78.39% F1 score: 75.17% Accuracy: 78.39 %	Precision: 68.15% Recall: 70.01% F1 score: 67.69% Accuracy: 70.01%	

Table 4.10: Classification Algorithm Performance Metrics for Dataset 2

Algorithms	Support Vector Machines (SVM)	Decision Trees (DT)	Random Forest Classifier (RFC)
Confusion Matrix	[0 658 0 555 474] [302 540 1 646 0] [32 0 1738 69 23] [71 1 594 286 40] [0 0 264 9 897]	[0 0 0 1 1610] [63 1336 0 0 0] [354 349 936 0 106] [247 90 241 0 368] [0 0 0 0 1199]	[0 0 0 0 1472] [54 1212 0 0 0] [0 0 1594 14 4] [221 1 0 631 0] [0 0 3 12 1082]
Evaluation Metrics	Precision: 57.67% Recall: 54.65% F1 score: 54.10% Accuracy: 54.65%	Precision: 64.34% Recall: 73.63% F1 score: 67.12% Accuracy: 73.63%	Precision: 95.65% Recall: 95.09% F1 score: 95.01% Accuracy: 95.09%
Algorithms	Naive Bayes (NB)	k-Nearest Neighbors (k-NN)	
Confusion Matrix	[779 854 0 49 0] [39 1420 0 0 0] [0 0 1778 35 8] [98 242 637 14 0] [0 0 18 0 1229]	[0 0 0 0 1750] [1 1522 0 0 0] [0 0 1897 1 2] [7 0 0 1027 0] [0 0 1 0 1292]	
Evaluation Metrics	Precision: 68.96% Recall: 72.5% F1 score: 67.12% Accuracy: 72.5%	Precision: 99.84% Recall: 99.83% F1 score: 99.83% Accuracy: 99.83%	

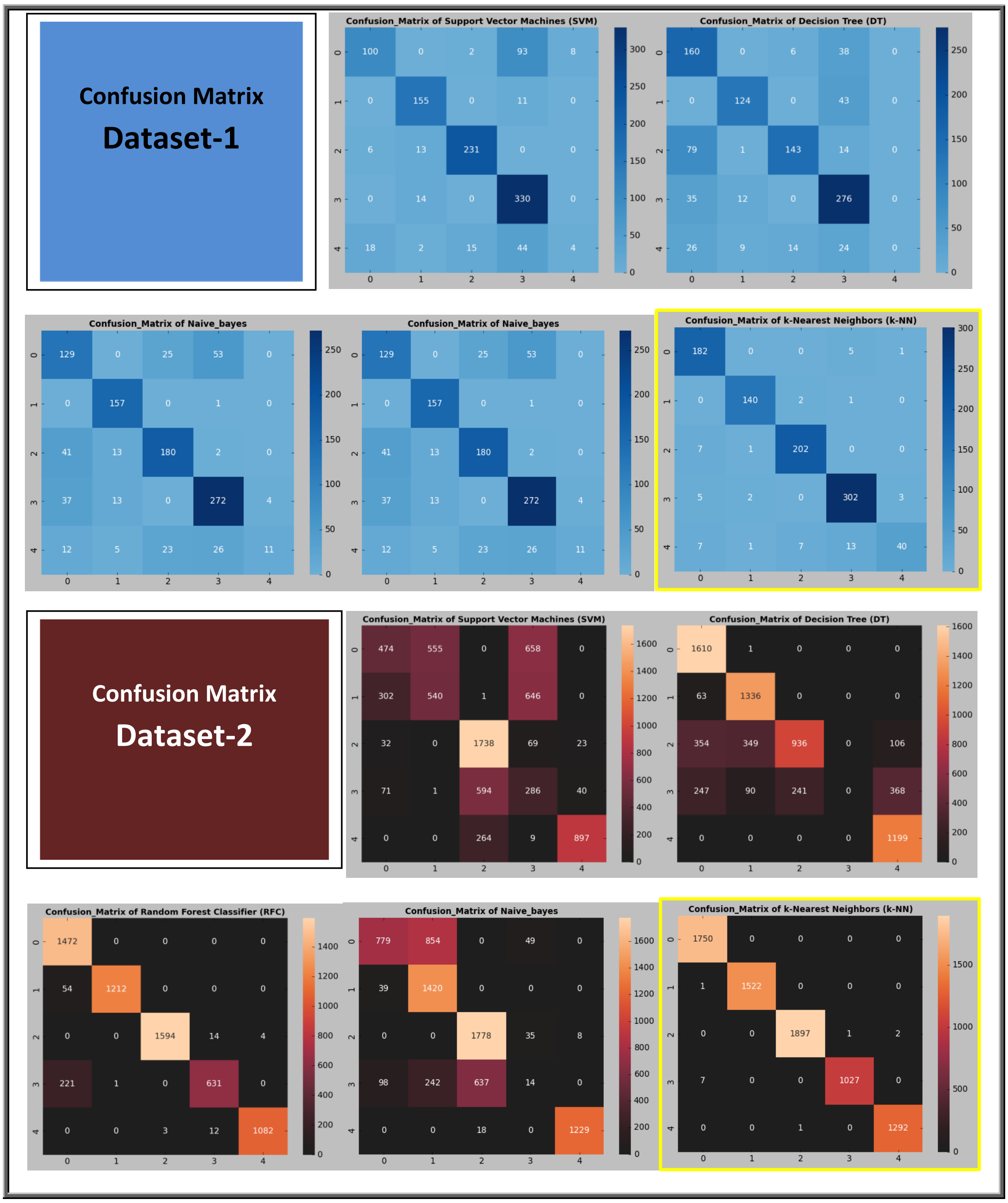


Figure 4.7: Confusion Matrix Comparison of Classification Algorithms in Datasets 1 & 2

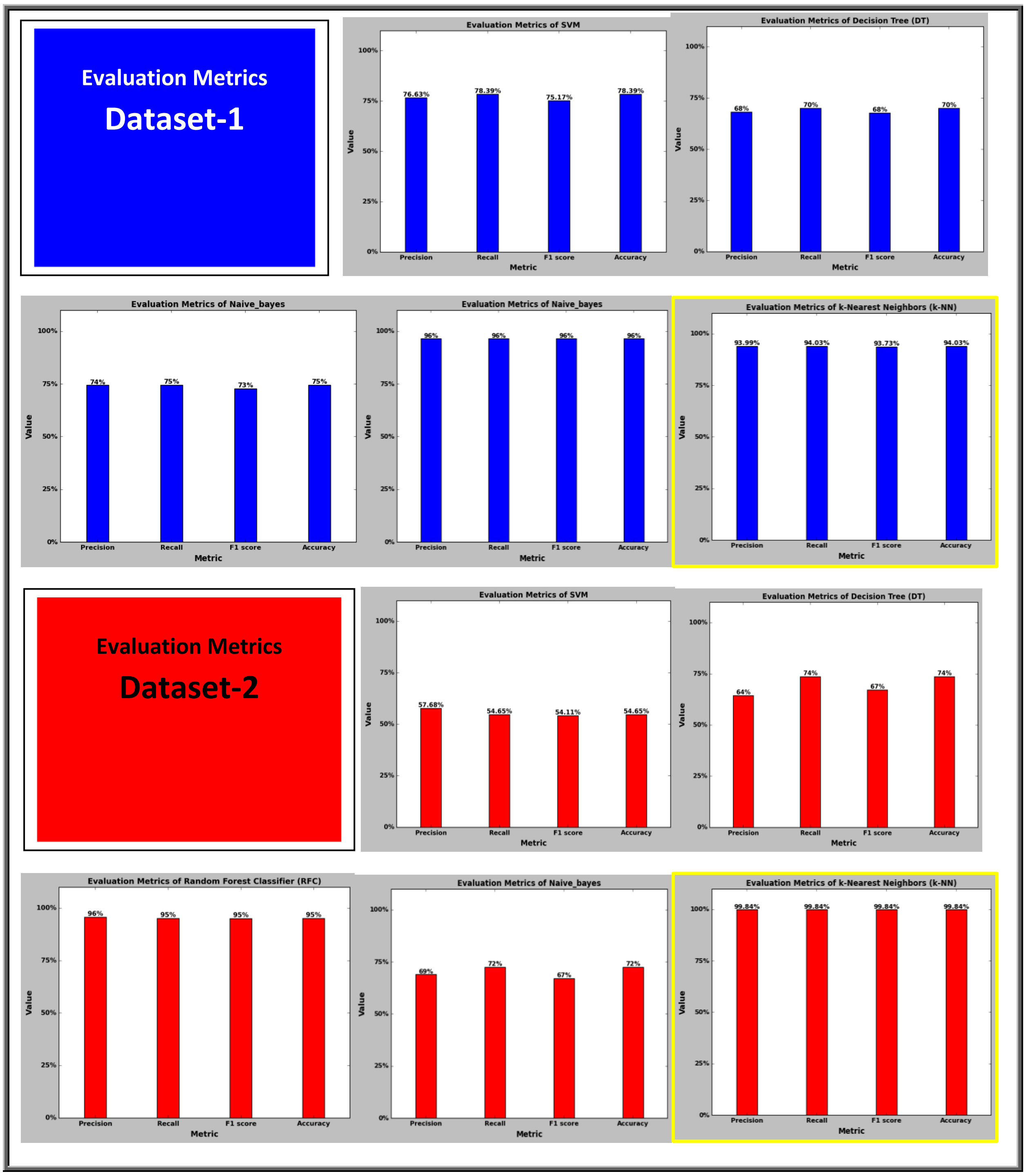


Figure 4.8: Evaluation Metrics Comparison of Classification Algorithms in Datasets 1 & 2

4.7 Evaluating the Queue Model

The dissertation's primary objective is to investigate the problem of load balancing among various priorities generated by the classification model, employing the queue model to handle this problem. The approach mentioned above has shown its efficacy in guaranteeing the best possible distribution of workloads and dealing with the significant issue of packet starvation. The dissertation proposes implementing a Multilevel Queue (MLQ) model as a more effective alternative to a single queue with five queues (0, 1, 2, 3, and 4). The algorithm (3.3) deals with five priorities stored in a distinct queue. Implementing this strategic shift has effectively improved the management of priorities, leading to significant enhancements in network performance and the quality of service (QoS).

4.7.1 The Results of Multilevel Queue (MLQ) Model

A hierarchical structure controls the transmission of tasks within the order of task prioritization. Tasks assigned a priority level of 0 continuously receive full consideration and are treated as the highest priority. After completing all tasks with a priority level of 0, tasks with a priority level of 1 are given priority. The aforementioned sequential methodology persists until all tasks are executed according to their priority.

However, this prioritization strategy posed challenges for lower-priority tasks, particularly those categorized as priorities 1, 2, 3 and 4, often leading to starvation. Such issues translated into delays in transmitting low-priority data and an overall decline in network performance. The present dissertation proposes a critical solution, the threshold concept, by determining a unique threshold value for every priority level. Once a specific threshold value is reached, lower-priority tasks are quickly assigned. By assigning priority 1 to a threshold value of 8, they are assigning priority 2 to a threshold value of 9, priority 3 to a threshold value of 10, and finally assigning priority 4 to a threshold value of 11. In the beginning, seven packets will be sent from priority zero. Until packet eight is reached, it will be sent to priority 1, then priority 2, 3, and 4, then it comes back and calls seven to send them to priority 0, and so on. If priority 0 ends, seven packets with priority 1 will be sent. The outcomes in Figures 4.9, 4.10, and 4.11 illustrate this strategy.

Furthermore, the operation of pushing and popping is depicted in Figure 4.9, which displays ten requests. In addition, Figure 4.10 offers valuable insights into the abovementioned procedures since it visually presents 20 requests. Figure 4.11 provides a more comprehensive analysis of these activities, specifically focusing on 100 requests. These figures provide significant visual representations that aid in comprehending the dynamic processes of pushing and popping within the queue model. Additionally, it explained the threshold concept and how it facilitates allocating work based on priorities. The strategic utilization of threshold values has shown significant efficacy in mitigating packet starvation and improving network efficiency.

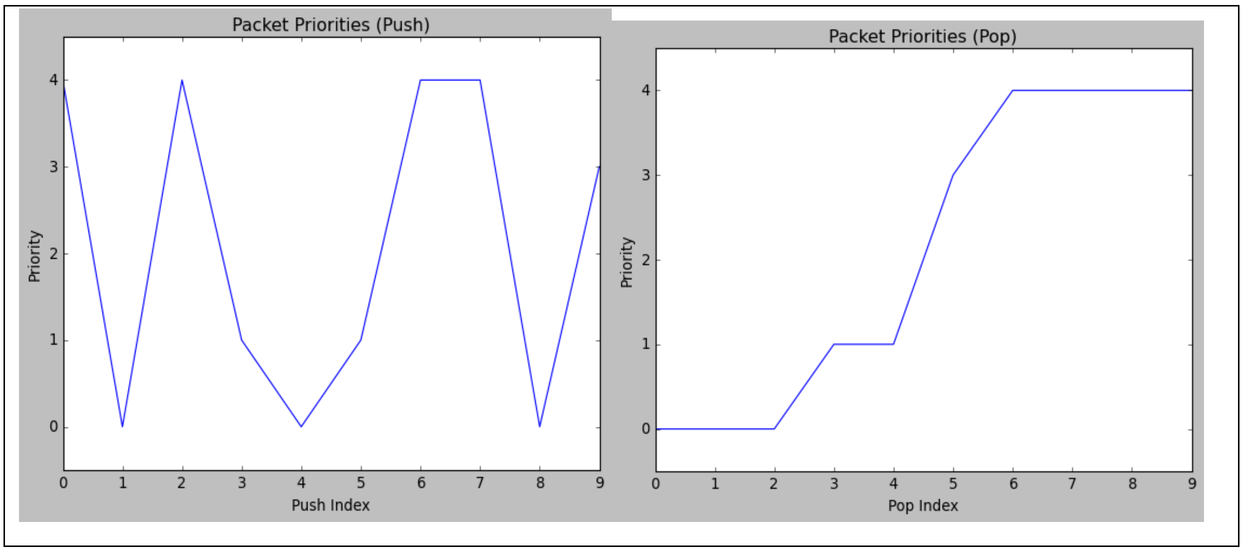


Figure 4.9: Pushing and Popping: Handling 10 Requests in MLQ

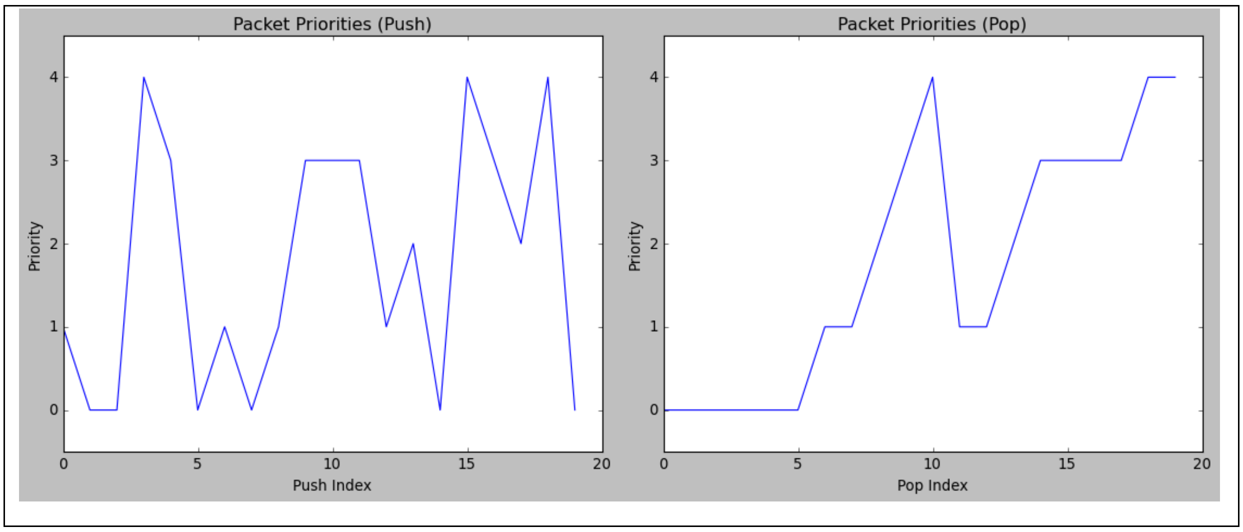


Figure 4.10: Exploring Push and Pop Operations with 20 Requests in MLQ

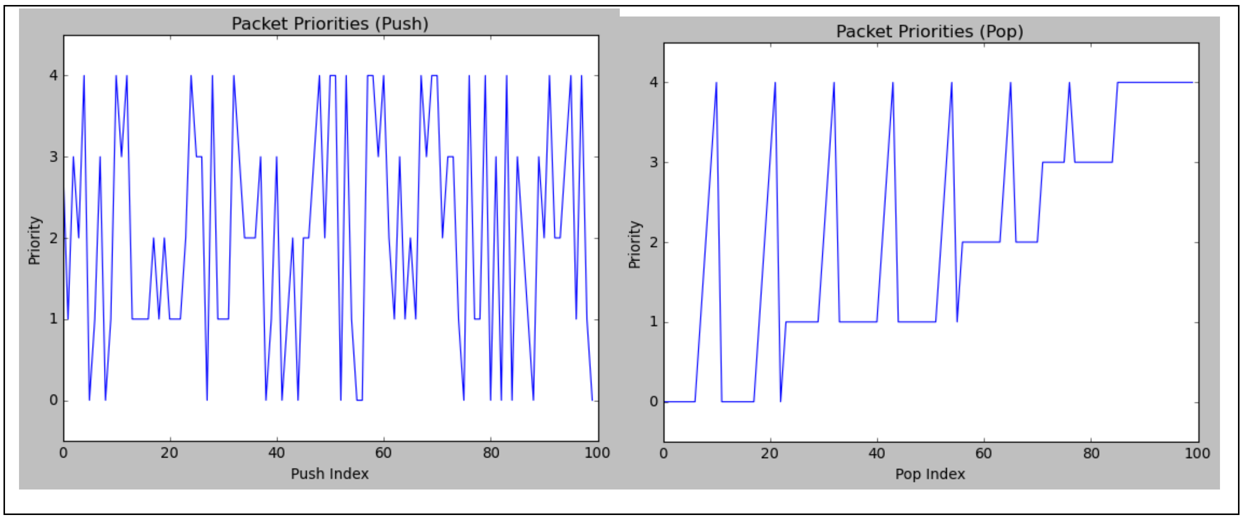
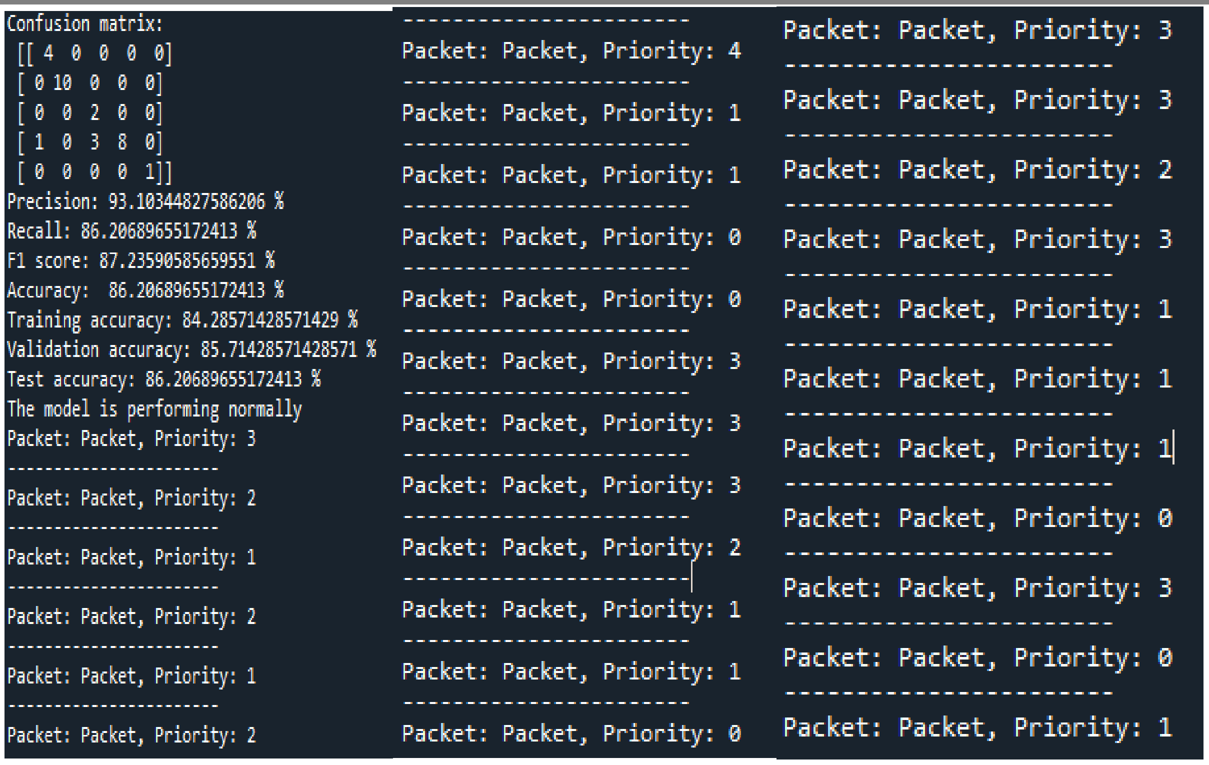


Figure 4.11: In-Depth Analysis: Managing 100 Requests through Push and Pop in MLQ

4.7.2 The Results of KNN Model with Queue Model

In the previous stage, the queue model was tested by sending varying numbers of packets (10, 20, and 100) and evaluating a push-and-pop operation. The current stage focuses on testing the results by combining two algorithms or modules, specifically the Machine Learning-MultiLevel Queue (KNN-MLQ) algorithm, as described in algorithm 3.4.

Figures 4.12, 4.13, and 4.14 illustrate the evaluation and testing of the KNN-MLQ model using the original dataset. The dataset consists of 100 rows selected from the larger dataset, which contains 30,000 rows. The test set was created by setting the test_size parameter to 0.30. The model was then used to predict 29 requests or rows, and each was assigned a priority value ranging from 0 to 4. These predictions are then passed to a queue model and stored using the push operation. The priority values are retrieved using the pop operation. Specifically, priority 0 is recalled seven times initially, and once the threshold 1 is reached, priority 1 is called. This process continues until threshold 2 is reached, at which point priority 2 is called. The same pattern follows for threshold 3, priority 3, threshold 4, and priority 4. Finally, the process returns to calling priority 0, repeating this sequence seven times. Sometimes, the call priority is limited to invoked 2 or 3 times, contingent upon the total number of priority 0 stored in a queue.

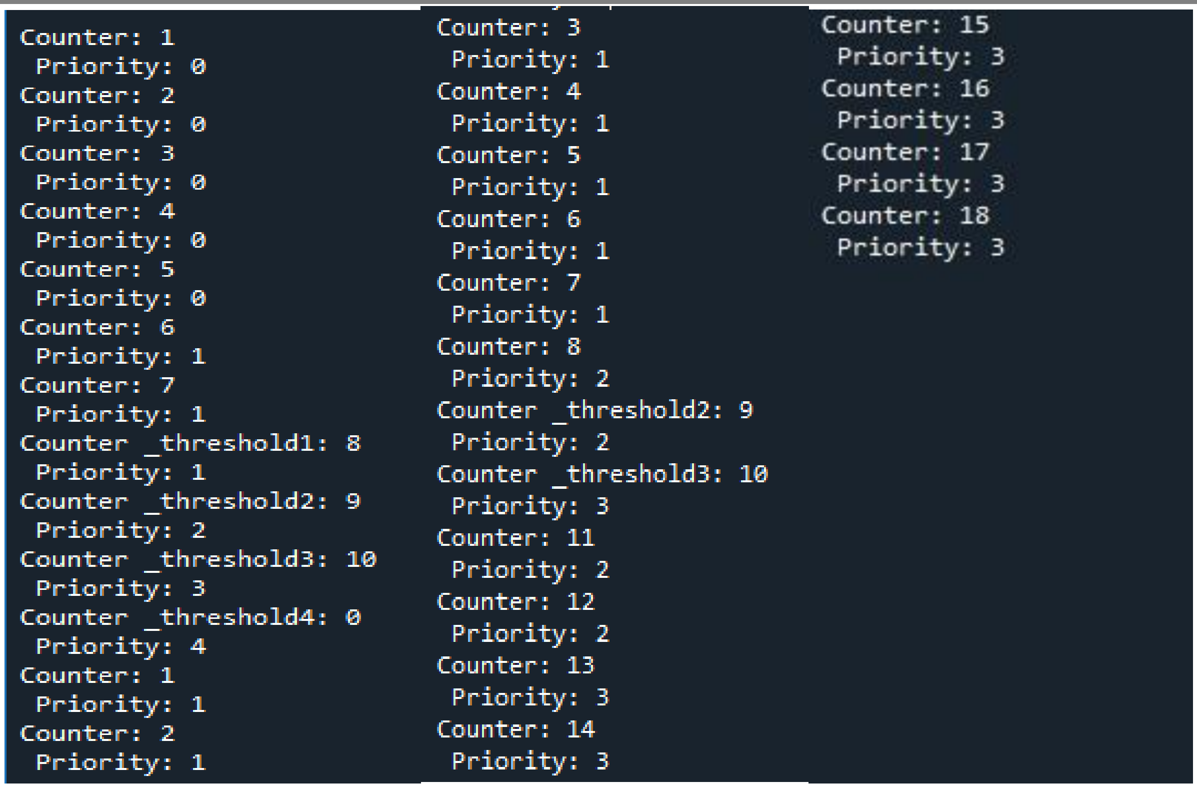


```
Confusion matrix:
 [[ 4  0  0  0  0]
 [ 0 10  0  0  0]
 [ 0  0  2  0  0]
 [ 1  0  3  8  0]
 [ 0  0  0  0  1]]
Precision: 93.10344827586206 %
Recall: 86.20689655172413 %
F1 score: 87.23590585659551 %
Accuracy:  86.20689655172413 %
Training accuracy: 84.28571428571429 %
Validation accuracy: 85.71428571428571 %
Test accuracy: 86.20689655172413 %
The model is performing normally
Packet: Packet, Priority: 3
---------------------
Packet: Packet, Priority: 2
---------------------
Packet: Packet, Priority: 1
---------------------
Packet: Packet, Priority: 2
---------------------
Packet: Packet, Priority: 1
---------------------
Packet: Packet, Priority: 2
```

```
-----------------------
Packet: Packet, Priority: 4
-----------------------
Packet: Packet, Priority: 1
-----------------------
Packet: Packet, Priority: 1
-----------------------
Packet: Packet, Priority: 0
-----------------------
Packet: Packet, Priority: 0
-----------------------
Packet: Packet, Priority: 3
-----------------------
Packet: Packet, Priority: 3
-----------------------
Packet: Packet, Priority: 3
-----------------------
Packet: Packet, Priority: 2
-----------------------
Packet: Packet, Priority: 1
-----------------------
Packet: Packet, Priority: 1
-----------------------
Packet: Packet, Priority: 0
```

```
Packet: Packet, Priority: 3
-----------------------
Packet: Packet, Priority: 3
-----------------------
Packet: Packet, Priority: 2
-----------------------
Packet: Packet, Priority: 3
-----------------------
Packet: Packet, Priority: 1
-----------------------
Packet: Packet, Priority: 1
-----------------------
Packet: Packet, Priority: 1
-----------------------
Packet: Packet, Priority: 0
-----------------------
Packet: Packet, Priority: 3
-----------------------
Packet: Packet, Priority: 0
-----------------------
Packet: Packet, Priority: 1
```

Figure 4.12: Testing KNN-MLQ Model with Push Operation



```
Counter:  1
 Priority:  0
Counter:  2
 Priority:  0
Counter:  3
 Priority:  0
Counter:  4
 Priority:  0
Counter:  5
 Priority:  0
Counter:  6
 Priority:  1
Counter:  7
 Priority:  1
Counter _threshold1:  8
 Priority:  1
Counter _threshold2:  9
 Priority:  2
Counter _threshold3:  10
 Priority:  3
Counter _threshold4:  0
 Priority:  4
Counter:  1
 Priority:  1
Counter:  2
 Priority:  1
```

```
Counter:  3
 Priority:  1
Counter:  4
 Priority:  1
Counter:  5
 Priority:  1
Counter:  6
 Priority:  1
Counter:  7
 Priority:  1
Counter:  8
 Priority:  2
Counter _threshold2:  9
 Priority:  2
Counter _threshold3:  10
 Priority:  3
Counter:  11
 Priority:  2
Counter:  12
 Priority:  2
Counter:  13
 Priority:  3
Counter:  14
 Priority:  3
```

```
Counter:  15
 Priority:  3
Counter:  16
 Priority:  3
Counter:  17
 Priority:  3
Counter:  18
 Priority:  3
```

Figure 4.13: Testing KNN-MLQ Model with Pop Operation

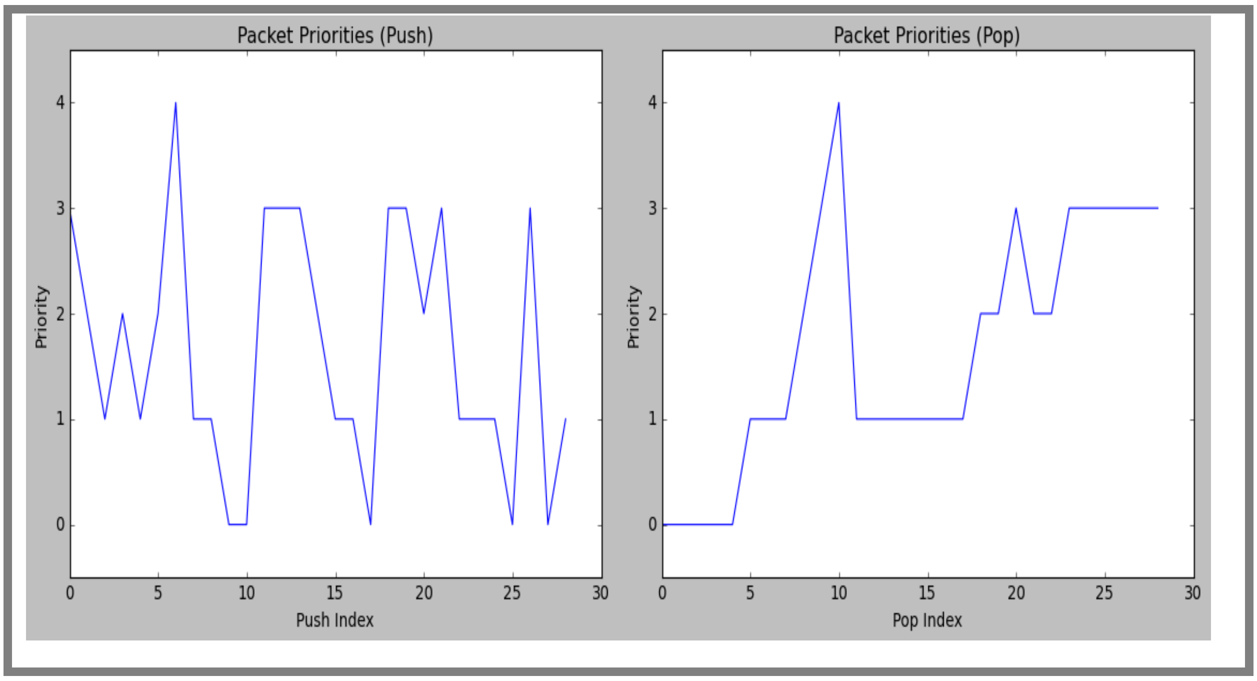


Figure 4.14: Pushing and Popping: Efficient Handling of 29 Requests in the KNN-MLQ

4.8 Evaluating the Load Balancing Scheduling (LBS)

Load-balancing algorithms are classified as static, dynamic, and hybrid. Static load balancing algorithms such as Random (R), Round Robin (RR), and Weight Round Robin (WRR) distribute workloads uniformly without considering real-time system conditions. However, dynamic load balancing algorithms, such as the Least Connection (LC) approach, distribute workloads based on real-time system conditions to optimize performance. This dissertation introduces a new taxonomy for hybrid load-balancing algorithms based on dynamic and static algorithms.

Two algorithms proposed and developed in this dissertation are based on dynamic load-balancing scheduling and distribute workloads based on real-time system conditions to optimize performance. The first is Least Resource Load (LRL), and the second is Hybrid Least Connection and Least Resource Load (LCLRL). LRL calculates CPU and memory use. In contrast, the algorithm balances load using the fewest connections, CPU, and memory. These six algorithms will be implemented in an environment, emulator, and network topology based on the requirements in the next section 4.8.1 .

4.8.1 Proposal of Emulator and Network Topology

The percentage distribution of emulation platforms utilized in the SDN Network, as mentioned in Chapter 1, Section 1.6.3, Figure 1.5. Table 4.11 displays the percentage of emulator platforms. The present study proposes the utilization of Mininet, an emulator program widely used in SDN networks, which also obtained the highest percentage, which is 50%, compared to the remainder of the software ecosystem.

Table 4.11: Percentage of the Essential Software Ecosystem used in the SDN Network

Software Ecosystem	Mininet	C and C++	Matlab	NS3,NS2	Python	CloudSim	Others
Percentage	50%	10%	5%	5%	7.5%	5%	17.5%

Mininet, a well-known open-source network emulator, is widely used to simulate and test SDN environments. SDN is a new networking architecture that efficiently separates the control plane from the data plane, enabling centralized network management and programmability. The primary design of Mininet is to be used for operating systems based on the Linux kernel. However, running Mininet on Windows using virtualization platforms such as VirtualBox may cause additional complexity. The utilization of Windows may not provide the same advantages in terms of performance and compatibility when compared to directly executing a Linux-based operating system.

Hence, the current study strongly recommends implementing and utilizing Mininet on a Linux operating system directly, and The version used is Linux Ubuntu 20.04.4 LTS. To optimize performance and achieve good results during the implementation of the suggested model, ensure that the computer components are of high quality. Table 4.12 overviews the computer components required for this specific purpose.

Table 4.12: Overviews the Computer Components

Parameters	OS	CPU	Ram	HDD	Graphics	OS Type
Details	Linux Ubuntu 20.04.4 LTS	Intel Corei7-3625QM 2.2GHz *8	8 GB	512GB SSD	AMD/ Intel4000	64-bit

The current study proposes a network topology known as a flat topology, specifically a single-switch topology, a network configuration consisting of only one network switch. All devices are connected to a single switch, and the type is an open virtual switch, known as OVS, which also includes a single SDN controller, specifically the POX-Controller. POX is primarily written in the Python programming language, and it is easy to use and has rapid development capabilities. An open-flow protocol establishes connections between the controller and switch, and other components are three hosts (H1, H2, and H3) and three servers (S1, S2, and S3) , his type of web browsing. Table 4.13 overviews the topology components and specifications of the emulator. Figure 4.15 illustrates the allocation of IP addresses across all devices within the network design and suggested topology..

Table 4.13: Overviews the Topology Components and Emulator specifications

Parameters	Controller architecture	Type of Controller	Number of Switch & Type	Number of servers & type	Number of Clients	Southbound interface
Component	Single controller	POX	1, OVS	3, Web Servers	3	OpenFlow 1.0

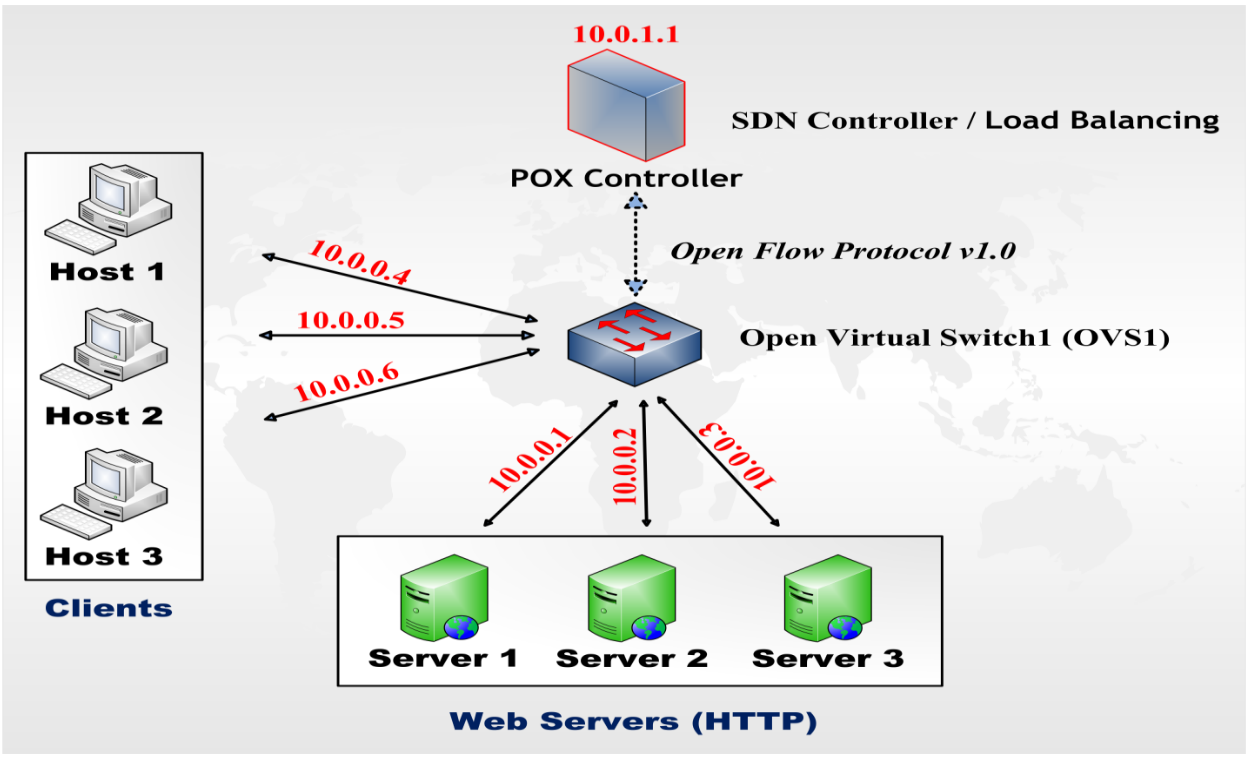


Figure 4.15: Proposed Network Topology

4.8.2 Evaluation Metrics in Load Balancing Scheduling Algorithms

The results from Chapter 1, Figure 1.3, which has 30 parameters, and Figure 1.4, which has eight important parameters, show that the dissertation recommends using the four most important parameters to evaluate load-balancing scheduling algorithms. These four parameters are response time, latency, throughput, and the degree of load balancing. The response time can be calculated using equations 2.8 and 2.9, with a lower value indicating a more desirable result. Similarly, the latency can be calculated using equations 2.10 and 2.11, with a lower value indicating a more desirable result.

Additionally, the throughput can be evaluated using equation 2.12, where a higher value signifies a better outcome. The degree of LB can be computed using equations 2.13 to 2.18, with the highest value indicating the best result. The selection of the optimal algorithm in statics and dynamics load balancing that achieves the best results across all factors will be based on evaluating the results obtained from these parameters.

4.8.3 The Results of Load Balancing Scheduling

The parameters are considered fundamental metrics to evaluate algorithmic results. This evaluation happens by applying four algorithms (R, RR, WRR, and LC) and two proposed algorithms (LRL and LCLRL), which are based on algorithm (3.4) and algorithm (3.5). The algorithms should be implemented in the Linux environment using the Mininet simulation. Subsequently, the proposed network, which is a flat network, should be established and connected to the POX controller. Static files will be used in the generation of traffic.

Additionally, as indicated in Table 4.2, two distinct types of traffic will be generated. Type 1 traffic involves the creation of files manually, encompassing five files of varying sizes (1 Mbyte, 10 Mbyte, 25 Mbyte, 50 Mbyte, and 75 Mbyte). On the other hand, Type 2 traffic consists of real files sourced from Facebook, containing 16 files with sizes ranging from 0.1 Kbyte to less than 10 Mbyte. These files were manually generated to reflect the ratios of real files.

By comprehensively analyzing several performance parameters such as response time, latency, throughput, and load balancing, the initial step involves evaluating and selecting the most optimal algorithm among the three static algorithms, R, RR, and WRR, based on the high-parameter results. The second step involves selecting the most suitable algorithm from three dynamic algorithms: LC, proposed LRL, and Proposed LCLRL, based on the high parameter results between them. Subsequently, the optimal dynamic algorithm will be employed with the proposed model. Table 4.14 presents the statistical analysis of response time parameters for the algorithms used in datasets 1 and 2.

Table 4.14: Statistics of Response Time Parameter for Algorithms - Datasets 1 and 2

Dataset 1						
	Static Algorithms			Dynamic Algorithms		
No. of Request	Random	RR	WRR	LC	LRL	LCLRL
300	0.044	0.042	0.050	0.047	0.047	0.050
3,000	0.049	0.050	0.048	0.049	0.048	0.050
15,000	0.049	0.049	0.049	0.049	0.049	0.050
45,000	0.049	0.050	0.050	0.049	0.048	0.050
Enhancement	%4.51	%4.50	%1.50	3%	4%	0.50%
Dataset 2						
No. of Request	Random	RR	WRR	LC	LRL	LCLRL
3,000	0.0120	0.0129	0.0124	0.0126	0.0120	0.0125
15,000	0.0128	0.0133	0.0128	0.0139	0.0135	0.0139
45,000	0.0131	0.0136	0.0141	0.0140	0.0140	0.0146
90,000	0.0135	0.0141	0.0138	0.0137	0.0139	0.0144
Enhancement	%7.85	%3.31	%4.84	2.82%	4.36%	0.77%

The results of the LBS method were computed for the response time parameter, as indicated in Table 4.14. The random algorithm demonstrated the highest improvement value among the static algorithms, achieving 4.51% in dataset 1 and 7.85% in dataset 2. In comparison, the RR and WRR algorithms achieved 4.50% and 1.50% in dataset 1 and 3.31% and 4.84% in dataset 2, respectively. Among the dynamic algorithms, the LRL algorithm displayed the best performance, achieving 4% in dataset 1 and 4.36% in dataset 2. In contrast, the LC and LCLRL algorithms achieved 3% and 0.50% in dataset 1 and 2.82% and 0.77% in dataset 2, respectively. Figure 4.16 represent the comparison between algorithms based on response time in datasets 1 and 2.

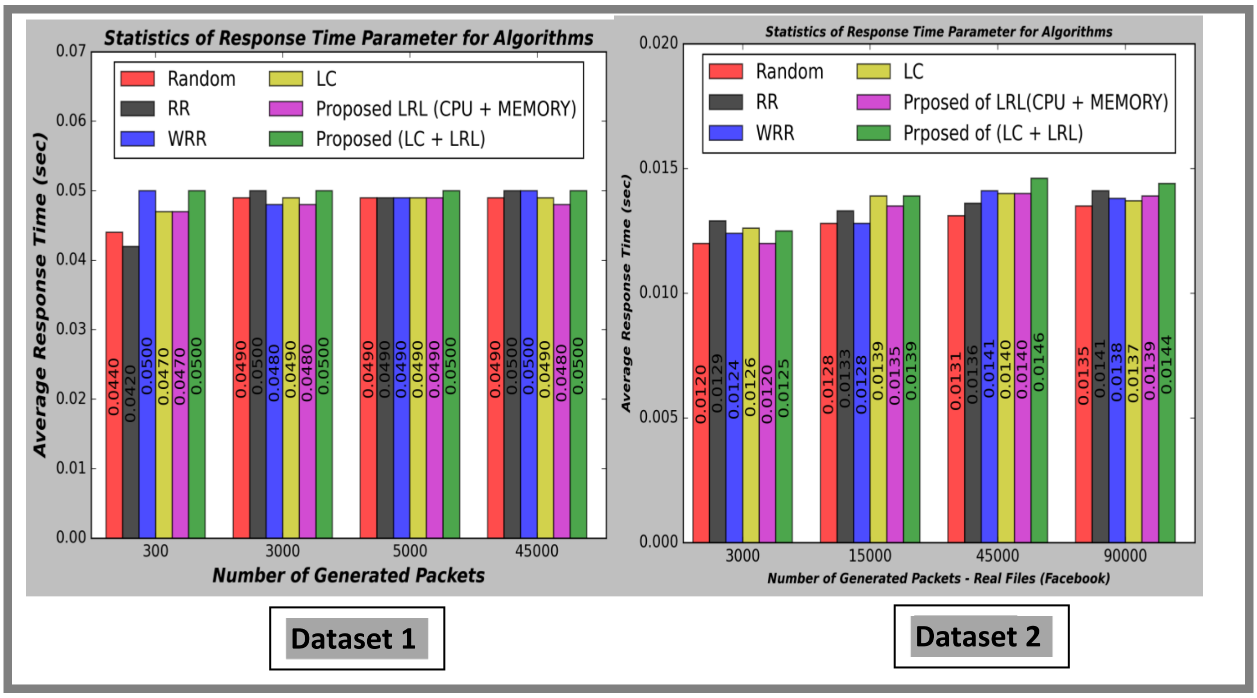


Figure 4.16: Statistics of Response Time Parameter for Algorithms in Datasets 1 and 2

Table 4.15 presents the statistical analysis of latency parameters for the algorithms used in datasets 1 and 2.

Table 4.15: Statistics of Latency Parameter for Algorithms - Datasets 1 and 2

Dataset 1						
	Static Algorithms			Dynamic Algorithms		
No. of Request	Random	RR	WRR	LC	LRL	LCLRL
300	0.040	0.040	0.046	0.044	0.043	0.046
3,000	0.044	0.045	0.043	0.045	0.043	0.045
15,000	0.044	0.045	0.045	0.046	0.045	0.045
45,000	0440.	0.045	0.045	0.044	0.044	0.045
Enhancement	%5.45	%3.80	%1.65	1.64%	3.84%	0.54%
Dataset 2						
No. of Request	Random	RR	WRR	LC	LRL	LCLRL
3,000	0.0054	0.0061	0.0058	0.0057	0.0053	0.0056
15,000	0.0059	0.0065	0.0062	0.0065	0.0062	0.0064
45,000	0.0060	0.0064	0.0066	0.0066	0.0065	0.0069
90,000	0.0063	0.0065	0.0064	0.0063	0.0063	0.0068
Enhancement	%10.27	%2.91	%4.94	4.56%	7.72%	2.43%

The LBS approach estimated the latency parameter, as presented in Table 4.15. The random method exhibited the most significant performance improvement compared to the static algorithms, with an optimization of 5.45%

in dataset 1 and 10.27% in dataset 2. When comparing the performance of the RR and WRR algorithms, it was seen that the RR method achieved an accuracy rate of 3.80% in dataset 1 and 2.91% in dataset 2. On the other hand, the WRR algorithm achieved an accuracy rate of 1.65% in dataset 1 and 4.94% in dataset 2. The LRL algorithm demonstrated the best results among the dynamic algorithms, with an accuracy rate of 3.84% in dataset 1 and 7.72% in dataset 2. Through comparison, the LC and LCLRL algorithms attained accuracy rates of 1.64% and 0.54% in dataset 1 and 4.56% and 2.43% in dataset 2, respectively. Figure 4.17 depict a comparative analysis of algorithms for latency in datasets 1 and 2.

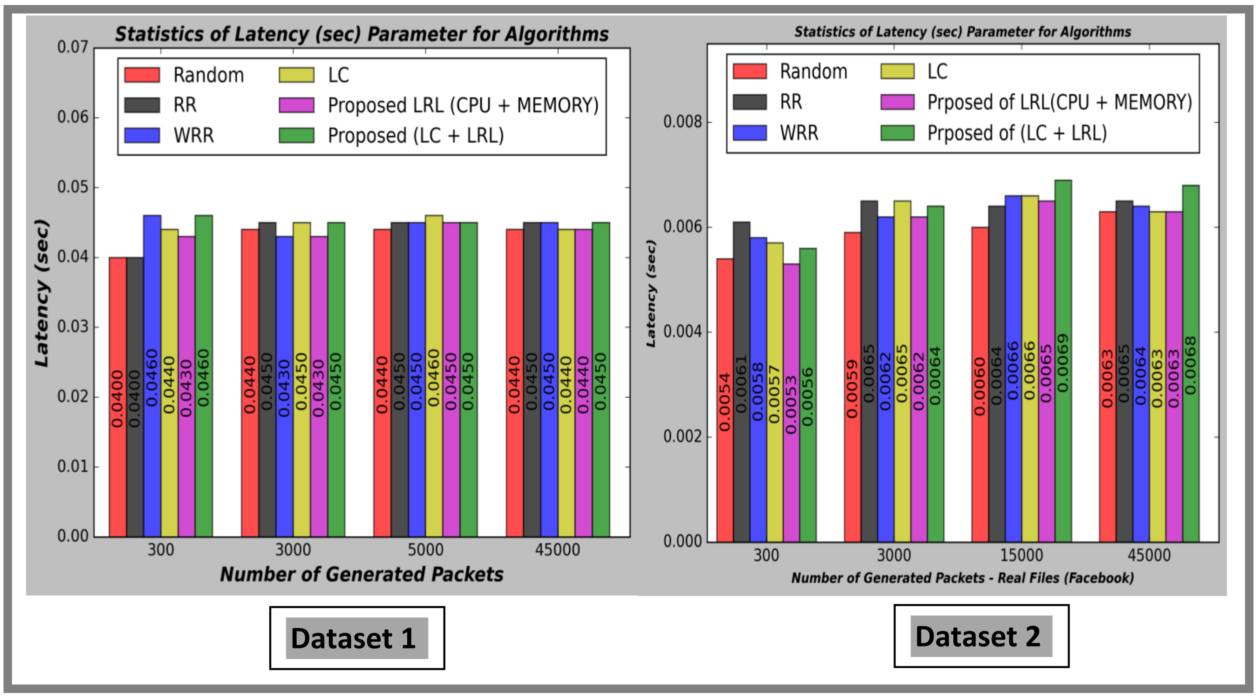


Figure 4.17: Statistics of Latency Parameter for Algorithms in Datasets 1 and 2

Table 4.16 presents the statistical analysis of throughput parameters for the algorithms used in datasets 1 and 2.

Table 4.16: Statistics of Throughput Parameter for Algorithms - Datasets 1 and 2

Dataset 1						
	Static Algorithms			Dynamic Algorithms		
No. of Request	Random	RR	WRR	LC	LRL	LCLRL
300	0.2261	0.2346	0.1998	0.2111	0.2090	0.1965
3,000	0.0203	0.0145	0.0207	0.0202	0.0207	0.0198
15,000	0.0041	0.0040	0.0040	0.0040	0.0040	0.0040
45,000	0.0013	0.0013	0.0013	0.0013	0.0014	0.0013
Enhancement	%6.29	%6.36	%5.64	5.91%	5.87%	5.54%
Dataset 2						
No. of Request	Random	RR	WRR	LC	LRL	LCLRL
3,000	0.0829	0.0779	0.0814	0.0793	0.0828	0.0801
15,000	0.0156	0.0149	0.0153	0.0146	0.0147	0.0143
45,000	0.0050	0.0049	0.0047	0.0047	0.0047	0.0045
90,000	0.0024	0.0023	0.0024	0.0024	0.0023	0.0023
Enhancement	%26.48	%25.00	%25.95	25.25%	26.12%	25.30%

The results of the LBS method were computed for the throughput parameter, as indicated in Table 4.16. The RR algorithm demonstrated the highest improvement value among the static algorithms in dataset 1, achieving 6.36%; in comparison, the R and WRR algorithms achieved 6.29% and 5.64%, respectively. While The R algorithm demonstrated the highest improvement value among the static algorithms in dataset 2, achieving 26.48%, the WRR and RR algorithms achieved 25.95% and 25.00%, respectively. Among the dynamic algorithms, the LC algorithm displayed the best performance in dataset 1, achieving 5.91%; in comparison, the LRL and LCLRL algorithms achieved 5.87% and 5.54%, respectively. While the LRL algorithm displayed the best performance in dataset 2, achieving 26.12%, in comparison, the LCLRL and LC algorithms achieved 25.30% and 25.25%, respectively. Figure 4.18 compare algorithms based on throughput requests per second (RPS) in datasets 1 and 2.

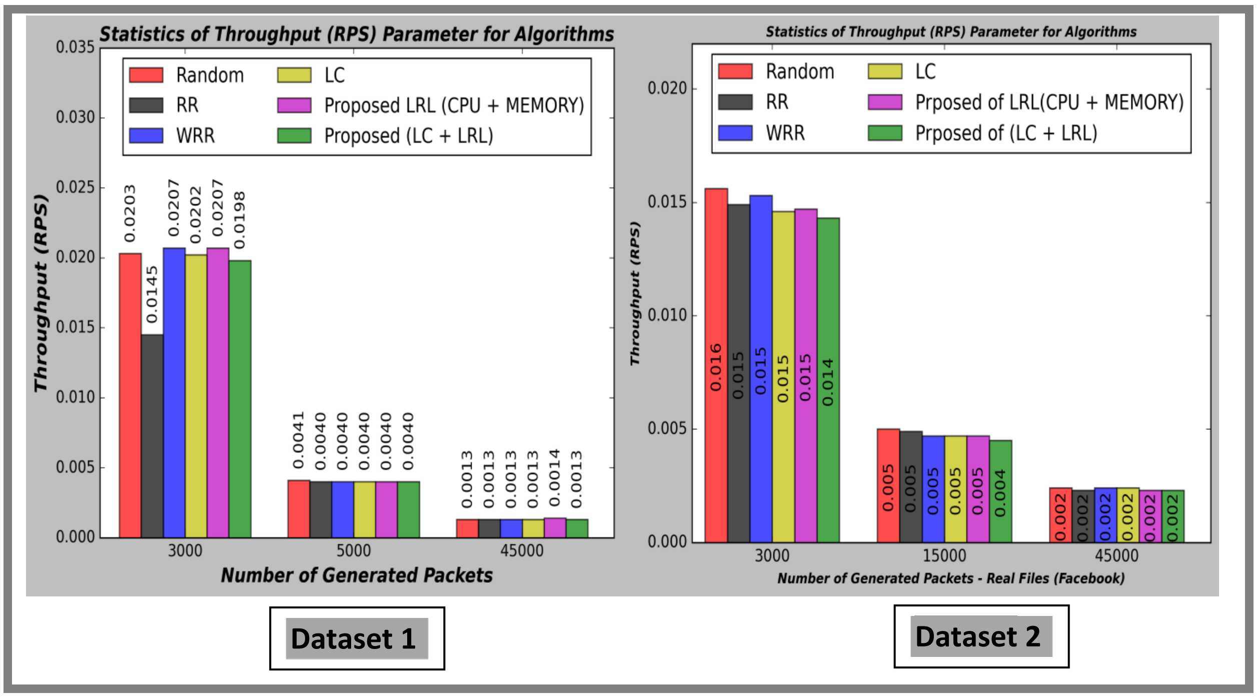


Figure 4.18: Statistics of Throughput Parameter for Algorithms in Datasets 1 and 2

Table 4.17 presents the statistical analysis of degree of load balancing parameters for the algorithms used in datasets 1 and 2.

Table 4.17: Statistics of Degree of LB Parameter for Algorithms - Datasets 1 and 2

Dataset 1						
	Static Algorithms			Dynamic Algorithms		
No. of Request	Random	RR	WRR	LC	LRL	LCLRL
300	0.246	0.227	0.284	0.301	0.281	0.297
3,000	0.236	0.266	0.256	0.239	0.234	0.244
15,000	0.243	0.198	0.250	0.237	0.239	0.241
45,000	0.239	0.238	0.233	0.234	0.244	0.249
Enhancement	%24.10	%23.22	%25.57	25.27%	24.95%	25.77%
Dataset 2						
No. of Request	Random	RR	WRR	LC	LRL	LCLRL
3,000	0.327	0.257	0.280	0.345	0.285	0.299
15,000	0.230	0.171	0.155	0.226	0.274	0.215
45,000	0.268	0.194	0.222	0.194	0.216	0.192
90,000	0.268	0.142	0.216	0.223	0.262	0.191
Enhancement	%27.32	%19.10	%21.82	24.70%	25.92%	22.43%

The results of the LBS method were computed for the degree of load balancing parameter, as indicated in Table 4.17. The WRR algorithm demonstrated the highest improvement value among the static algorithms in dataset 1, achieving 25.57%; in comparison, the R and RR algorithms achieved

24.10% and 23.22%, respectively. While The R algorithm demonstrated the highest improvement value among the static algorithms in dataset 2, achieving 26.48%, the WRR and RR algorithms achieved 25.95% and 25.00%, respectively. Among the dynamic algorithms, the LCLRL algorithm displayed the best performance in dataset 1, achieving 25.77%; in comparison, the LC and LRL algorithms achieved 25.27% and 24.95%, respectively. While the LRL algorithm displayed the best performance in dataset 2, achieving 25.92%, in comparison, the LC and LCLRL algorithms achieved 24.70% and 22.43%, respectively. Figure 4.19 compare algorithms based on the degree of load balancing in datasets 1 and 2.

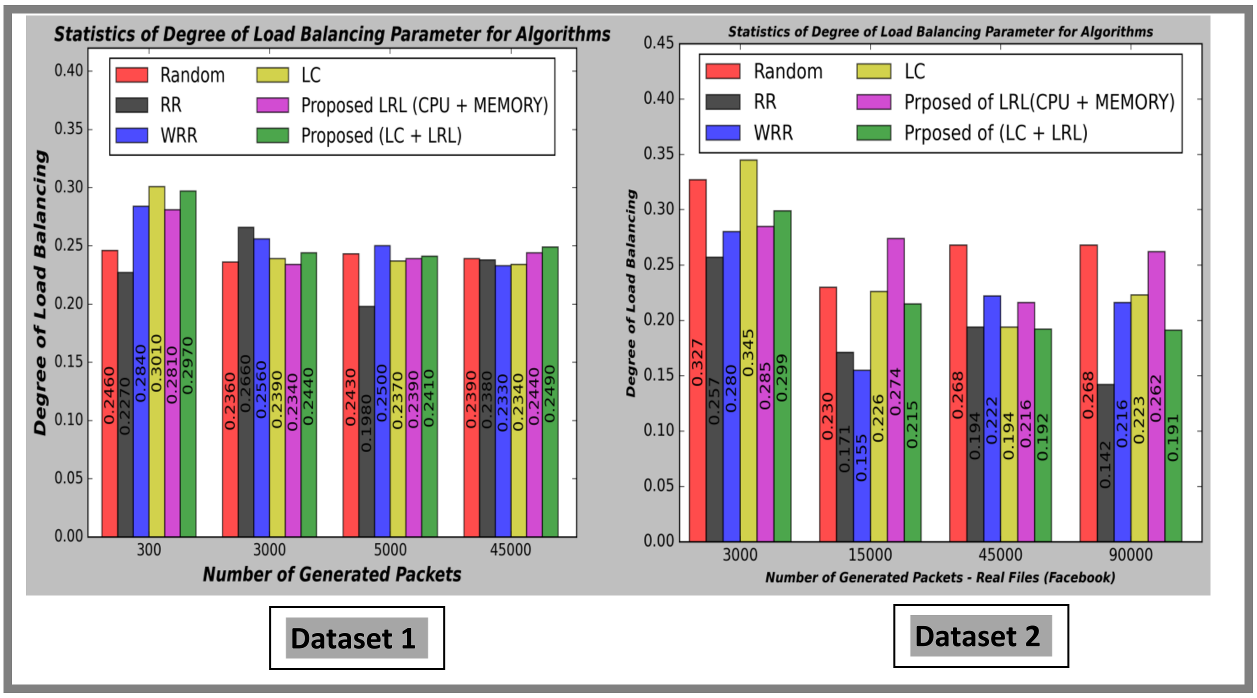


Figure 4.19: Statistics of Degree of LB Parameter for Algorithms in Datasets 1 and 2

These procedures help to determine and discover that the proposed LRL algorithm consistently outperforms other dynamic algorithms, as demonstrated through the evaluation and analysis of the parameters utilized and the final assessments achieved. This helps identify the optimal dynamic algorithm. The LRL algorithm shows the highest evaluation scores for factors such as response time and latency and achieved excellent performance in Dataset 1. Moreover, in

Dataset 2, the LRL algorithm displays the highest percentages and overall evaluation across all parameters. Consequently, the above method will be implemented and applied to the proposed model.

4.9 Evaluating the Proposed Model

The evaluation of the proposed model algorithm integrates three techniques: Machine Learning with Multilevel Queue and Load Balancing Scheduling (ML-MLQLBS). This integration aims to create an intelligent model known as KNN-MLQLRL. Algorithm 3.6 describes the phases of the suggested model, which aims to enhance intelligent dynamic load balancing. A comparative analysis will be conducted with six existing load balancing algorithms (R, RR, WRR, LC, LRL, LCLRL) to evaluate the success and efficiency of the proposed model. This evaluation will involve the calculation of four key performance indicators: response time, latency, throughput, and degree of load balancing. Subsequently, the outcomes will be compared with the other algorithms' outcomes to determine the best in datasets (1 and 2).

4.9.1 The Results of Proposed Model

The proposed model will be implemented in a Linux environment using the Mininet simulation. Subsequently, the model is run inside the POX controller and deployed in the flat network architecture, employed to generate traffic, his type dynamic files and content to 6 features within the range of 0 and 1, utilizing a random distribution approach. The suggested model would receive packets containing eight distinct features from clients, initiating the subsequent stages of classification and prediction to determine the cluster type and priority category. Subsequently, the model will transmit the priority to the queuing model. Subsequently, employing the suggested algorithm for selecting the optimal server based on the least resource load (LRL) algorithm, The server receives the priority in addition to the first packet's contents, previously sent from clients that include the features.

Table 4.18 presents the statistical analysis of degree of load balancing parameters for the algorithms used in datasets 1 and 2.

Table 4.18: Statistics of Response Time Parameter for Algorithms - Datasets 1 and 2

Dataset 1							
No. of Request	Random	RR	WRR	LC	LRL	LCLRL	Model (KNN-MLQLRL)
300	0.044	0.042	0.050	0.047	0.047	0.050	0.0054
3,000	0.049	0.050	0.048	0.049	0.048	0.050	0.0099
15,000	0.049	0.049	0.049	0.049	0.049	0.050	0.0113
45,000	0.049	0.050	0.050	0.049	0.048	0.050	0.0115
Enhancement	%4.51	%4.50	%1.50	3%	4%	0.50%	80.95%
Dataset 2							
No. of Request	Random	RR	WRR	LC	LRL	LCLRL	Model (KNN-MLQLRL)
3,000	0.0120	0.0129	0.0124	0.0126	0.0120	0.0125	0.0093
15,000	0.0128	0.0133	0.0128	0.0139	0.0135	0.0139	0.0114
45,000	0.0131	0.0136	0.0141	0.0140	0.0140	0.0146	0.0116
90,000	0.0135	0.0141	0.0138	0.0137	0.0139	0.0144	0.0117
Enhancement	%7.85	%3.31	%4.84	2.82%	4.36%	0.77%	21.3%

The LBS methods and proposed model results for the response time parameter are indicated in Table 4.18. The proposed model (KNN-MLQLRL) demonstrated the highest improvement value in datasets 1 and 2, with dataset 1 achieving 25.57% and other algorithms, the Random, RR, LRL, LC, WRR, and LCLRL algorithms achieved 4.51%, 4.50%, 4%, 3%, 1.50%, and 0.50%, respectively. While dataset 2 achieved 21.3% and other algorithms, the Random, WRR, LRL, RR, LC, and LCLRL algorithms achieved 7.85%, 4.84%, 4.36%, 3.31%, 2.82%, and 0.77%, respectively. Figure 4.20 compare algorithms based on response time in datasets 1 and 2.

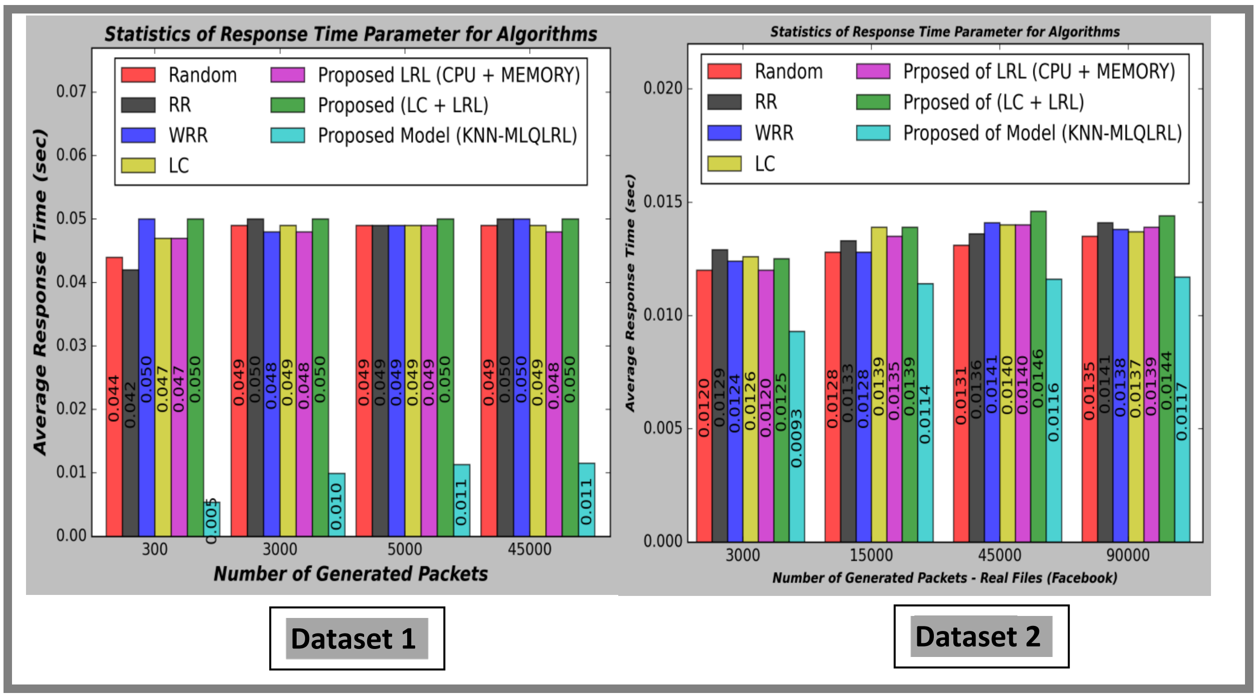


Figure 4.20: Statistics of Response Time Parameter for Algorithms in Datasets 1 and 2

Table 4.19 presents the statistical analysis of latency parameters for the algorithms used in datasets 1 and 2.

Table 4.19: Statistics of Latency Parameter for Algorithms - Datasets 1 and 2

Dataset 1							
No. of Request	Random	RR	WRR	LC	LRL	LCLRL	Model (KNN-MLQLRL)
300	0.040	0.040	0.046	0.044	0.043	0.046	0.0006
3,000	0.044	0.045	0.043	0.045	0.043	0.045	0.0007
15,000	0.044	0.045	0.045	0.046	0.045	0.045	0.0008
45,000	0440.	0.045	0.045	0.044	0.044	0.045	0.0008
Enhancement	%5.45	%3.80	%1.65	1.64%	3.84%	0.54%	98.40%
Dataset 2							
No. of Request	Random	RR	WRR	LC	LRL	LCLRL	Model (KNN-MLQLRL)
3,000	0.0054	0.0061	0.0058	0.0057	0.0053	0.0056	0.0008
15,000	0.0059	0.0065	0.0062	0.0065	0.0062	0.0064	0.0008
45,000	0.0060	0.0064	0.0066	0.0066	0.0065	0.0069	0.0008
90,000	0.0063	0.0065	0.0064	0.0063	0.0063	0.0068	0.0009
Enhancement	%10.27	%2.91	%4.94	4.56%	7.72%	2.43%	87.43%

Table 4.19 displays the LBS methods and the outcomes of the suggested model for the latency parameter. The KNN-MLQLRL model displays the most substantial improvement in datasets 1 and 2, with an accuracy of 98.40% in dataset 1. In comparison, the Random, LRL, RR, WRR, LC, and LCLRL algorithms achieved accuracies of 5.45%, 3.84%, 3.80%, 1.65%, 1.64%, and 0.54%, respectively. Dataset 2 displays a performance of 87.43%, whereas the Random, LRL, WRR, LC, RR, and LCLRL algorithms have success rates of 10.27%, 7.72%, 4.94%, 4.56%, 2.91%, and 2.43%, respectively. This analysis focuses on the performance of algorithms in datasets 1 and 2, specifically on latency. Figure 4.21 present a comparative evaluation of these algorithms.

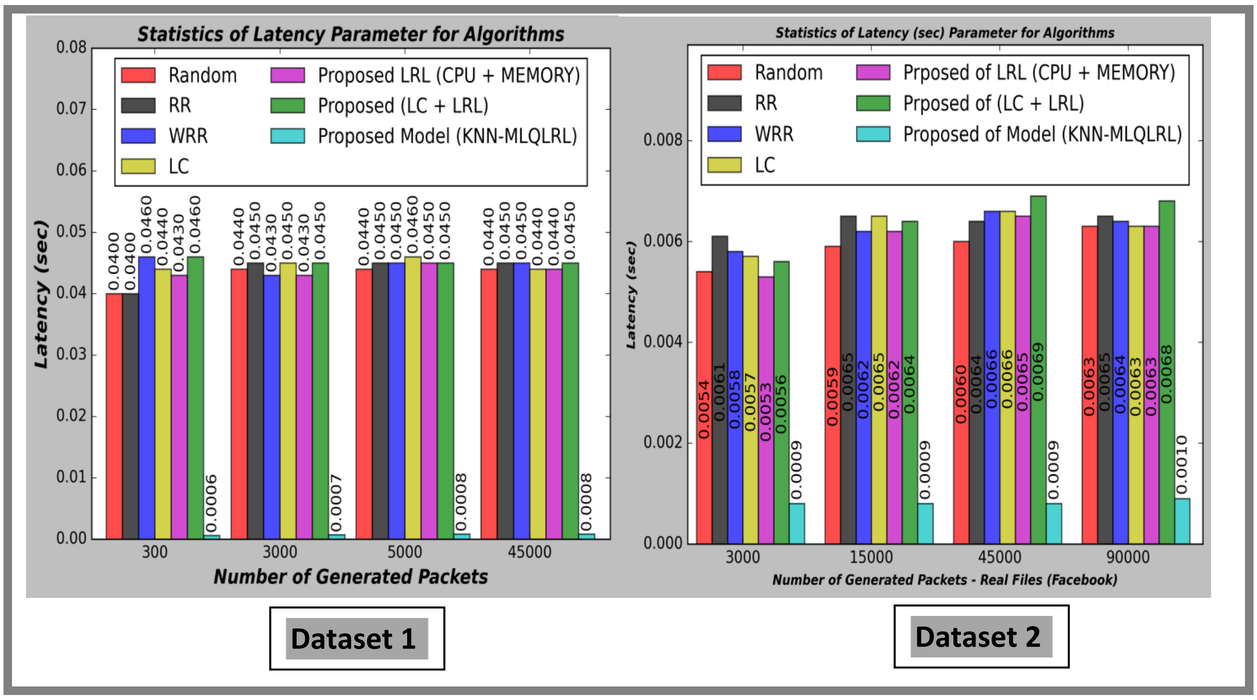


Figure 4.21: Statistics of Latency Parameter for Algorithms in Datasets 1 and 2

Table 4.20 presents the statistical analysis of throughput parameters for the algorithms used in datasets 1 and 2.

Table 4.20: Statistics of Throughput Parameter for Algorithms - Datasets 1 and 2

Dataset 1							
No. of Request	Random	RR	WRR	LC	LRL	LCLRL	Model (KNN-MLQLRL)
300	0.2261	0.2346	0.1998	0.2111	0.2090	0.1965	0.8560
3,000	0.0203	0.0145	0.0207	0.0202	0.0207	0.0198	0.1000
15,000	0.0041	0.0040	0.0040	0.0040	0.0040	0.0040	0.0177
45,000	0.0013	0.0013	0.0013	0.0013	0.0014	0.0013	0.0057
Enhancement	%6.29	%6.36	%5.64	5.91%	5.87%	5.54%	24.48%
Dataset 2							
No. of Request	Random	RR	WRR	LC	LRL	LCLRL	Model (KNN-MLQLRL)
3,000	0.0829	0.0779	0.0814	0.0793	0.0828	0.0801	0.1081
15,000	0.0156	0.0149	0.0153	0.0146	0.0147	0.0143	0.0174
45,000	0.0050	0.0049	0.0047	0.0047	0.0047	0.0045	0.0058
90,000	0.0024	0.0023	0.0024	0.0024	0.0023	0.0023	0.0028
Enhancement	%26.48	%25.00	%25.95	25.25%	26.12%	25.30%	33.52%

Table 4.20 presents the LBS approaches employed and the corresponding outputs of the proposed model about the throughput parameter. The KNN-MLQLRL model demonstrates the most significant enhancement in datasets 1 and 2, achieving an accuracy of 24.48% in dataset 1. The RR, Random, LC, LRL, WRR, and LCLRL algorithms achieved accuracies of 6.36, 6.29, 5.91, 5.87, 5.64, and 5.54, respectively, when compared. In Dataset 2, the performance was determined to be 33.52%. In comparison, the Random, LRL, WRR, LCLRL, LC, and RR algorithms exhibited success rates of 26.48%, 26.12%, 25.95%, 25.3%, 25.25%, and 25.0%, respectively. The present analysis evaluates algorithmic performance within datasets 1 and 2, focusing on throughput. Figure 4.22 depict a comparison of the different algorithms.

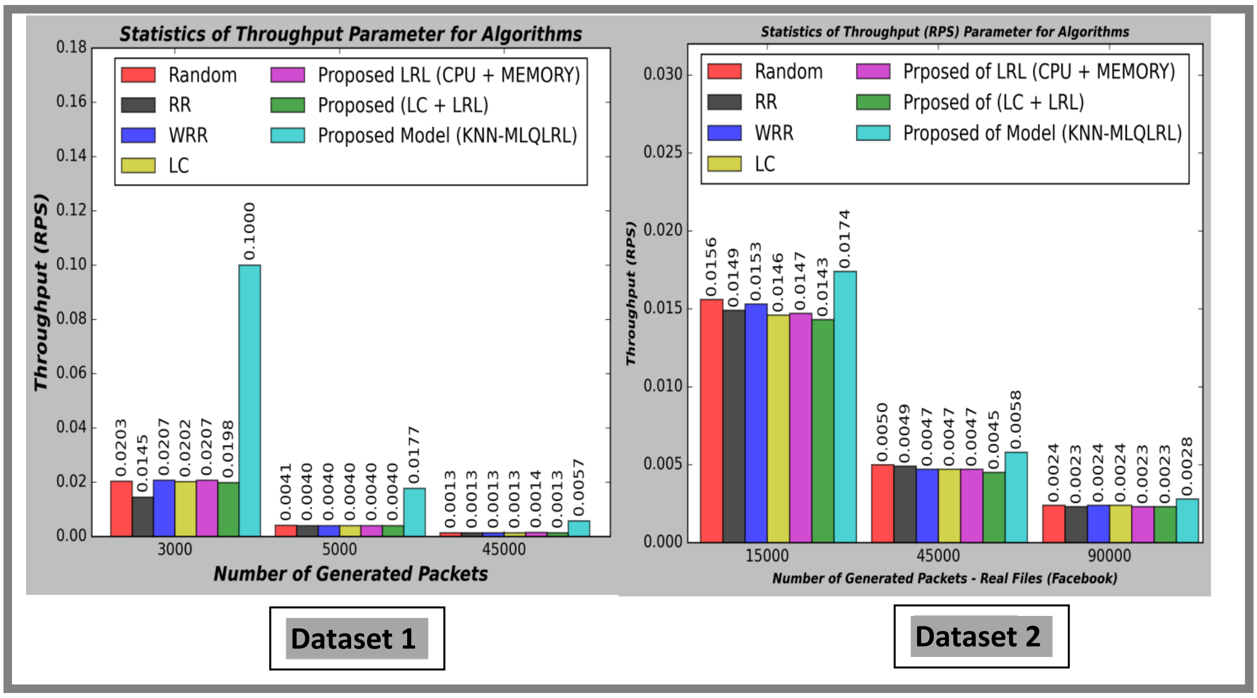


Figure 4.22: Statistics of Throughput Parameter for Algorithms in Datasets 1 and 2

Table 4.21 presents the statistical analysis of degree of load balancing parameters for the algorithms used in datasets 1 and 2.

Table 4.21: Statistics of Degree of LB Parameter for Algorithms - Datasets 1 and 2

Dataset 1							
No. of Request	Random	RR	WRR	LC	LRL	LCLRL	Model (KNN-MLQLRL)
300	0.246	0.227	0.284	0.301	0.281	0.297	0.815
3,000	0.236	0.266	0.256	0.239	0.234	0.244	0.750
15,000	0.243	0.198	0.250	0.237	0.239	0.241	0.690
45,000	0.239	0.238	0.233	0.234	0.244	0.249	0.624
Enhancement	%24.10	%23.22	%25.57	25.27%	24.95%	25.77%	71.97%
Dataset 2							
No. of Request	Random	RR	WRR	LC	LRL	LCLRL	Model (KNN-MLQLRL)
3,000	0.327	0.257	0.280	0.345	0.285	0.299	0.711
15,000	0.230	0.171	0.155	0.226	0.274	0.215	0.700
45,000	0.268	0.194	0.222	0.194	0.216	0.192	0.618
90,000	0.268	0.142	0.216	0.223	0.262	0.191	0.588
Enhancement	%27.32	%19.10	%21.82	24.70%	25.92%	22.43%	65.42%

Table 4.21 displays the LBS methods and the outcomes of the suggested model for the degree of load balancing parameter. The KNN-MLQLRL model displayed the most substantial improvement in datasets 1 and 2, with an accuracy of 71.97% in dataset 1. In comparison, the LCLRL, WRR, LC, LRL,

Random, and RR algorithms achieved accuracies of 25.77, 25.57, 25.27, 24.95, 24.1, and 23.22, respectively. Dataset 2 displayed a performance of 65.42%, whereas the Random, LRL, LC, LCLRL, WRR, and RR algorithms had success rates of 27.32, 25.92, 24.7, 22.43, 21.82, and 19.1, respectively. This analysis focuses on the performance of algorithms in datasets 1 and 2, specifically on degree of load balancing. Figure 4.23 present a comparative evaluation of these algorithms.

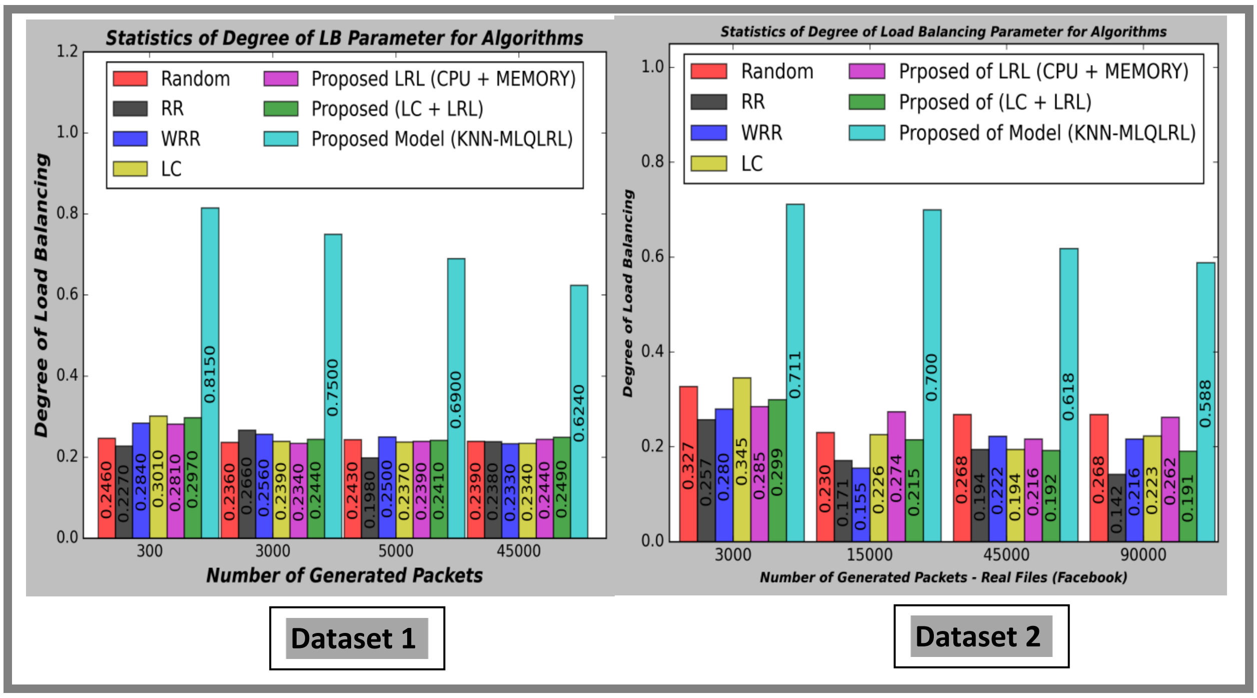


Figure 4.23: Statistics of Degree of LB Parameter for Algorithms in Dataset 1 and 2

Chapter Five

The Conclusions and Future Works

5.1 Conclusions

The primary objective of the proposed model is to enhance the Quality of Service (QoS) and overall network performance in the SDN network. In addition, this dissertation aims to identify optimal dynamic load-balancing algorithms that effectively minimize response time and latency while enhancing throughput and achieving a higher degree of load balancing. So, this dissertation contributes to the extended previous literature on load balancing in SDN. A survey of the latest advancements in the field is conducted, highlighting areas of research that require further investigation and identifying gaps among existing studies. The primary objective of this survey is to improve the understanding of load balancing challenges in SDN through an examination of LB-Algorithms, LB-Classification, LB-Techniques, and LB-Metrics.

The main conclusions of this dissertation, based on the development and implementation of the proposed model, are as follows:

1. There were two types of generating and creating data: manual generation and files of random sizes, and real files and sizes. According to the final results, the second type of file in the load-balancing distribution between servers was an actual distribution close to reality. In contrast, the first type was very different from the logical distribution.
2. In some situations, it is necessary to detect and identify noise in the data because of its significance and influence on the dataset and then handle it. This dissertation proposes a method that combines two algorithms, namely OPTICS and GMM. Optics is particularly effective at splitting data points into clusters and identifying noise. The second algorithm employs a probabilistic distribution to handle noise and redistribute it to neighboring clusters. This approach demonstrated better noise management and achieved the most balanced allocation of data points among clusters compared to alternative algorithms such as K-means, DBSCAN, OPTICS, and GMM.

3. Overall, the k-nearest neighbor classification algorithm outperforms other classification algorithms, such as (DT, SVM, NB, and RF) regarding the accuracy and performance metrics for both datasets.
4. The effectiveness of the proposed Multilevel Queue (MLQ) model is based on its capability to handle and organize priorities derived from the classification and prediction models. This model positively impacts network performance and Quality of Service (QoS) by reducing packet loss and delay by treating packet starvation using a threshold concept, wherein a distinct threshold value is determined for each priority level. Once a predetermined threshold is reached, tasks of lower priority are quickly assigned without waiting a long time to be process.
5. Regarding calculating the most critical resources for servers, for example, low CPU and memory usage in achieving load distribution has proven successful and more stable than relying on random or sequential distribution between servers or calculating the least connection. The dissertation proposed two algorithms that generally achieve high optimization on the principles of dynamic load-balancing scheduling between servers; the first is Least Resource Load (LRL), and the second is Hybrid Least Connection and Least Resource Load (LCLRL). The LRL algorithm calculates the lower CPU and memory usage. On the other hand, the LCLRL algorithm combines the least connection and the least CPU and memory usage to achieve load balancing. Compared with the four standard algorithms, the three algorithms are static, such as R, RR, and WRR, and one is the dynamic algorithm LC.

5.1 Future Work

The following are many possible future works that need consideration:

1. Studying an alternative method for traffic capture that involves utilizing deep packet inspection (DPI) as a pattern-matching approach instead of the statistical approach.
2. The proposed model can be implemented on other load balancing datasets.
3. Applying the proposed model to calculate other Metrics to improve QoS .
4. Studying the design of a program that captures packets, analyzes the features, and sends them to the classification model dynamically in real time.
5. Studying other methods of optimizing classification models, such as neural networks or deep learning algorithms
6. Studying alternative dynamic load balancing scheduling approaches to optimize Quality of Service (QoS). Specifically, it investigates calculating response time or latency inside servers instead of CPU and memory utilization metrics for improved load balancing.
7. Proposing a method that can identify critical files or features in the dataset and give special priority to them without waiting in the queue model.
8. Studying the development of the proposed model capable of receiving packets that include static files and dynamic features together.
9. Studying and applying methods for determining the optimal number of queues based on waiting time calculations and other parameters.
10. Using heuristic algorithms instead of a queue model.

References

References

[1] G. F. Huseien and K. W. Shah, "A review on 5G technology for smart energy management and smart buildings in Singapore," *Energy and AI,* vol. 7, p. 100116, 2022/01 2022.

[2] N. Khan, R. b. Salleh, A. Koubaa, Z. Khan, M. K. Khan, and I. Ali, "Data plane failure and its recovery techniques in SDN: A systematic literature review," *Journal of King Saud University - Computer and Information Sciences,* vol. 35, no. 3, pp. 176-201, 2023/03 2023.

[3] J. F. Balarezo, S. Wang, K. G. Chavez, A. Al-Hourani, and S. Kandeepan, "A survey on DoS/DDoS attacks mathematical modelling for traditional, SDN and virtual networks," *Engineering Science and Technology, an International Journal,* vol. 31, p. 101065, 2022/07 2022.

[4] S. M. A *et al.*, "A Survey on Routing Algorithms and Techniques Used to Improve Network Performance in Software-Defined Networking," presented at the 2023 2nd International Conference on Computational Systems and Communication (ICCSC), 2023/03/03, 2023. Available: http://dx.doi.org/10.1109/iccsc56913.2023.10143010

[5] M. Khalid, S. Hameed, A. Qadir, S. A. Shah, and D. Draheim, "Towards SDN-based smart contract solution for IoT access control," *Computer Communications,* vol. 198, pp. 1-31, 2023/01 2023.

[6] R. Deb and S. Roy, "A comprehensive survey of vulnerability and information security in SDN," *Computer Networks,* vol. 206, p. 108802, 2022/04 2022.

[7] M. Priyadarsini and P. Bera, "Software defined networking architecture, traffic management, security, and placement: A survey," *Computer Networks,* vol. 192, p. 108047, 2021/06 2021.

[8] A. Ahalawat, K. S. Babu, A. K. Turuk, and S. Patel, "A low-rate DDoS detection and mitigation for SDN using Renyi Entropy with Packet Drop," *Journal of Information Security and Applications,* vol. 68, p. 103212, 2022/08 2022.

[9] J. Ali, R. H. Jhaveri, M. Alswailim, and B.-h. Roh, "ESCALB: An effective slave controller allocation-based load balancing scheme for multi-domain SDN-enabled-IoT networks," *Journal of King Saud University - Computer and Information Sciences,* vol. 35, no. 6, p. 101566, 2023/06 2023.

[10] S. Chattopadhyaya and A. K. Sahoo, "Software defined networks: Current problems and future solutions," *Materials Today: Proceedings,* vol. 49, pp. 2989-2993, 2022.

[11] S. Zafar, Z. Lv, N. H. Zaydi, M. Ibrar, and X. Hu, "DSMLB: Dynamic switch-migration based load balancing for software-defined IoT network," *Computer Networks,* vol. 214, p. 109145, 2022/09 2022.

[12] P. Varalakshmi, M. A, N. B, R. P. G, and Y. S, "Intelligent load balancing in SDN," presented at the 2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS), 2022/03/25, 2022. Available: http://dx.doi.org/10.1109/icaccs54159.2022.9785334

[13] J. Zhang, X. Huang, J. Li, K. Xue, Q. Sun, and J. Lu, "A Dynamic Flow Table Management Method Based on Real-time Traffic Monitoring," presented at the 2022 IEEE 23rd International Conference on High Performance Switching and Routing (HPSR), 2022/06/06, 2022. Available: http://dx.doi.org/10.1109/hpsr54439.2022.9831366

[14] K. Boussaoud, M. Ayache, and A. En-Nouaary, "Performance Evaluation of Supervised ML Algorithms for Elephant Flow Detection in SDN," presented at the 2022 8th International

Conference on Optimization and Applications (ICOA), 2022/10/06, 2022. Available: http://dx.doi.org/10.1109/icoa55659.2022.9934652

[15] R. K. Chouhan, M. Atulkar, and N. K. Nagwani, "A framework to detect DDoS attack in Ryu controller based software defined networks using feature extraction and classification," *Applied Intelligence,* vol. 53, no. 4, pp. 4268-4288, 2022/06/07 2022.

[16] Y. Jang, N. Kim, and B.-D. Lee, "Traffic classification using distributions of latent space in software-defined networks: An experimental evaluation," *Engineering Applications of Artificial Intelligence,* vol. 119, p. 105736, 2023/03 2023.

[17] 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. 44932-44940, 2023.

[18] M. A. Gunavathie and S. Umamaheswari, "MLPRS: A Machine Learning-Based Proactive Re-Routing Scheme for flow classification and priority assignment," *Journal of Engineering Research,* p. 100075, 2023/04 2023.

[19] S. Begam, S. M, and S. N. R, "Load Balancing in DCN Servers through SDN Machine Learning Algorithm," ed: Research Square Platform LLC, 2021.

[20] A. Llorens-Carrodeguas, I. Leyva-Pupo, C. Cervelló-Pastor, L. Piñeiro, and S. Siddiqui, "An SDN-Based Solution for Horizontal Auto-Scaling and Load Balancing of Transparent VNF Clusters," (in eng), *Sensors (Basel, Switzerland),* vol. 21, no. 24, p. 8283, 2021.

[21] X. Zheng, W. Huang, H. Li, and G. Li, "Research on Generalized Intelligent Routing Technology Based on Graph Neural Network," *Electronics,* vol. 11, no. 18, p. 2952, 2022/09/17 2022.

[22] G. Kim, Y. Kim, and H. Lim, "Deep Reinforcement Learning-Based Routing on Software-Defined Networks," *IEEE Access,* vol. 10, pp. 18121-18133, 2022.

[23] H. Zheng, J. Guo, Q. Zhou, Y. Peng, and Y. Chen, "Application of improved ant colony algorithm in load balancing of software-defined networks," *The Journal of Supercomputing,* vol. 79, no. 7, pp. 7438-7460, 2022/11/27 2022.

[24] E. L. H. Bouzidi, A. Outtagarts, R. Langar, and R. Boutaba, "Dynamic clustering of software defined network switches and controller placement using deep reinforcement learning," *Computer Networks,* vol. 207, p. 108852, 2022/04 2022.

[25] H. Aldabbas, "Efficient bandwidth allocation in SDN-based peer-to-peer data streaming using machine learning algorithm," *The Journal of Supercomputing,* vol. 79, no. 6, pp. 6802-6824, 2022/11/15 2022.

[26] T. Malbašić, P. D. Bojović, Ž. Bojović, J. Šuh, and D. Vujošević, "Hybrid SDN Networks: A Multi-parameter Server Load Balancing Scheme," *Journal of Network and Systems Management,* vol. 30, no. 2, 2022/01/27 2022.

[27] H. Babbar, S. Parthiban, G. Radhakrishnan, and S. Rani, "A genetic load balancing algorithm to improve the QoS metrics for software defined networking for multimedia applications," *Multimedia Tools and Applications,* vol. 81, no. 7, pp. 9111-9129, 2022/01/08 2022.

[28] G. Rastogi and R. Sushil, "Analytical literature survey on existing load balancing schemes in cloud computing," presented at the 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), 2015/10, 2015. Available: http://dx.doi.org/10.1109/icgciot.2015.7380705

[29] A. A. Neghabi, N. Jafari Navimipour, M. Hosseinzadeh, and A. Rezaee, "Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature," *IEEE Access,* vol. 6, pp. 14159-14178, 2018.

[30] S. Afzal and G. Kavitha, "Load balancing in cloud computing – A hierarchical taxonomical classification," *Journal of Cloud Computing,* vol. 8, no. 1, 2019/12 2019.

[31] M. R. Belgaum, S. Musa, M. M. Alam, and M. M. Su'ud, "A Systematic Review of Load Balancing Techniques in Software-Defined Networking," *IEEE Access,* vol. 8, pp. 98612-98636, 2020.

[32] S.-N. Yang, C.-H. Ke, Y.-B. Lin, and C.-H. Gan, "Mobility management through access network discovery and selection function for load balancing and power saving in software-defined networking environment," *EURASIP Journal on Wireless Communications and Networking,* vol. 2016, no. 1, 2016/09/02 2016.

[33] W. Li *et al.*, "Reliability Assurance Dynamic SSC Placement Using Reinforcement Learning," *Information,* vol. 13, no. 2, p. 53, 2022/01/21 2022.

[34] P. Song, Y. Liu, T. Liu, and D. Qian, "Flow Stealer: lightweight load balancing by stealing flows in distributed SDN controllers," *Science China Information Sciences,* vol. 60, no. 3, 2017/01/23 2017.

[35] Y.-W. Ma, J.-L. Chen, Y.-H. Tsai, K.-H. Cheng, and W.-C. Hung, "Load-Balancing Multiple Controllers Mechanism for Software-Defined Networking," *Wireless Personal Communications,* vol. 94, no. 4, pp. 3549-3574, 2016/10/27 2016.

[36] N. Alzaben and D. W. Engels, "End-to-End Routing Algorithm Based on Max-Flow Min-Cut in SDN Controllers," presented at the 2022 24th International Conference on Advanced Communication Technology (ICACT), 2022/02/13, 2022. Available: http://dx.doi.org/10.23919/icact53585.2022.9728780

[37] X. Duan, A. M. Akhtar, and X. Wang, "Software-defined networking-based resource management: data offloading with load balancing in 5G HetNet," *EURASIP Journal on Wireless Communications and Networking,* vol. 2015, no. 1, 2015/06/23 2015.

[38] C. Wang, H. Ni, and L. Liu, "An Enhanced Message Distribution Mechanism for Northbound Interfaces in the SDN Environment," *Applied Sciences,* vol. 11, no. 10, p. 4346, 2021/05/11 2021.

[39] K. Kaur, S. Kaur, and V. Gupta, "Least Time Based Weighted Load Balancing Using Software Defined Networking," in *Communications in Computer and Information Science*, ed: Springer Singapore, 2017, pp. 309-314.

[40] S. Kaur and J. Singh, "Implementation of Server Load Balancing in Software Defined Networking," in *Advances in Intelligent Systems and Computing*, ed: Springer India, 2016, pp. 147-157.

[41] H. Gasmelseed and R. Ramar, "Traffic pattern-based load-balancing algorithm in software-defined network using distributed controllers," *International Journal of Communication Systems,* vol. 32, no. 17, p. e3841, 2018/11/08 2018.

[42] D. Franco, M. Aguado, and N. Toledo, "An Adaptable Train-to-Ground Communication Architecture Based on the 5G Technological Enabler SDN," *Electronics,* vol. 8, no. 6, p. 660, 2019/06/12 2019.

[43] T. Han and N. Ansari, "A Traffic Load Balancing Framework for Software-Defined Radio Access Networks Powered by Hybrid Energy Sources," *IEEE/ACM Transactions on Networking,* vol. 24, no. 2, pp. 1038-1051, 2016/04 2016.

[44] B. Kang and H. Choo, "An SDN-enhanced load-balancing technique in the cloud system," *The Journal of Supercomputing,* vol. 74, no. 11, pp. 5706-5729, 2016/12/07 2016.

[45] Y. Gao and T. Luo, "A load balancing scheme for supporting safety applications in heterogeneous software defined LTE-V networks," presented at the 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), 2017/10, 2017. Available: http://dx.doi.org/10.1109/pimrc.2017.8292478

[46] I. Ahammad, M. A. R. Khan, Z. U. Salehin, M. Uddin, and S. J. Soheli, "Improvement of QOS in an IoT ecosystem by integrating fog computing and SDN," *International Journal of Cloud Applications and Computing (IJCAC),* vol. 11, no. 2, pp. 48-66, 2021.

[47] L. Mamushiane and T. Shozi, "A QoS-based evaluation of SDN controllers: ONOS and OpenDayLight," in *2021 IST-Africa Conference (IST-Africa)*, 2021, pp. 1-10: IEEE.

[48] M. T. Islam, N. Islam, and M. A. Refat, "Node to Node Performance Evaluation through RYU SDN Controller," *Wireless Personal Communications,* vol. 112, no. 1, pp. 555-570, 2020/01/20 2020.

[49] X. Liu, L. Pan, C.-J. Wang, and J.-Y. Xie, "A Lock-Free Solution for Load Balancing in Multi-Core Environment," presented at the 2011 3rd International Workshop on Intelligent Systems and Applications, 2011/05, 2011. Available: http://dx.doi.org/10.1109/isa.2011.5873313

[50] H. M. Noman and M. N. Jasim, "A Proposed Dynamic Hybrid-Based Load Balancing Algorithm to Improve Resources Utilization in SDN Environment," in *International Conference on New Trends in Information and Communications Technology Applications*, 2021, pp. 147-162: Springer.

[51] M. Kumar and S. C. Sharma, "Dynamic load balancing algorithm for balancing the workload among virtual machine in cloud computing," *Procedia Computer Science,* vol. 115, pp. 322-329, 2017.

[52] Y. Fu *et al.*, "A Hybrid Hierarchical Control Plane for Flow-Based Large-Scale Software-Defined Networks," *IEEE Transactions on Network and Service Management,* vol. 12, no. 2, pp. 117-131, 2015/06 2015.

[53] M. Priyadarsini, J. C. Mukherjee, P. Bera, S. Kumar, A. H. M. Jakaria, and M. A. Rahman, "An adaptive load balancing scheme for software-defined network controllers," *Computer Networks,* vol. 164, p. 106918, 2019/12 2019.

[54] Y. Zhao, C. Liu, H. Wang, X. Fu, Q. Shao, and J. Zhang, "Load balancing-based multi-controller coordinated deployment strategy in software defined optical networks," *Optical Fiber Technology,* vol. 46, pp. 198-204, 2018.

[55] H. Zhong, Y. Fang, and J. Cui, "Reprint of "LBBSRT: An efficient SDN load balancing scheme based on server response time"," *Future Generation Computer Systems,* vol. 80, pp. 409-416, 2018.

[56] C. Li, Q. Cai, and Y. Lou, "Optimal data placement strategy considering capacity limitation and load balancing in geographically distributed cloud," *Future Generation Computer Systems,* vol. 127, pp. 142-159, 2022/02 2022.

[57] L. Boero, M. Cello, C. Garibotto, M. Marchese, and M. Mongelli, "BeaQoS: Load balancing and deadline management of queues in an OpenFlow SDN switch," *Computer Networks,* vol. 106, pp. 161-170, 2016/09 2016.

[58] Z. Guo *et al.*, "Improving the performance of load balancing in software-defined networks through load variance-based synchronization," *Computer Networks,* vol. 68, pp. 95-109, 2014/08 2014.

[59] J. Zhang, K. Xi, M. Luo, and H. J. Chao, "Load balancing for multiple traffic matrices using SDN hybrid routing," presented at the 2014 IEEE 15th International Conference on High

Performance Switching and Routing (HPSR), 2014/07, 2014. Available: http://dx.doi.org/10.1109/hpsr.2014.6900880

[60] A. Marini, "CHARACTER BUILDING THROUGH TEACHING LEARNING PROCESS: LESSON IN INDONESIA," *PONTE International Scientific Researchs Journal,* vol. 73, no. 5, 2017.

[61] A. K. Rangisetti, T. V. Pasca S, and B. R. Tamma, "QoS Aware load balance in software defined LTE networks," *Computer Communications,* vol. 97, pp. 52-71, 2017/01 2017.

[62] V. Borovskiy, J. Wust, C. Schwarz, W. Koch, and A. Zeier, "A linear programming approach for optimizing workload distribution in a cloud," *Cloud Computing,* pp. 127-132, 2011.

[63] H. Xu, X.-Y. Li, L. Huang, Y. Du, and Z. Liu, "Partial flow statistics collection for load-balanced routing in software defined networks," *Computer Networks,* vol. 122, pp. 43-55, 2017.

[64] D. B. L.D and P. Venkata Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Applied Soft Computing,* vol. 13, no. 5, pp. 2292-2303, 2013/05 2013.

[65] M. Moradi, M. Abbasi Dezfuli, and M. Hasan Safavi, "A new time optimizing probabilistic load balancing algorithm in grid computing," presented at the 2010 2nd International Conference on Computer Engineering and Technology, 2010. Available: http://dx.doi.org/10.1109/iccet.2010.5486187

[66] C. Chunling, L. Jun, and W. Ying, "An energy-saving task scheduling strategy based on vacation queuing theory in cloud computing," *Tsinghua Science and Technology,* vol. 20, no. 1, pp. 28-39, 2015/02 2015.

[67] A. Bhadani and S. Chaudhary, "Performance evaluation of web servers using central load balancing policy over virtual machines on cloud," presented at the Proceedings of the Third Annual ACM Bangalore Conference on - COMPUTE '10, 2010. Available: http://dx.doi.org/10.1145/1754288.1754304

[68] Y.-D. Lin, C. C. Wang, Y.-J. Lu, Y.-C. Lai, and H.-C. Yang, "Two-tier dynamic load balancing in SDN-enabled Wi-Fi networks," *Wireless Networks,* vol. 24, no. 8, pp. 2811-2823, 2017/04/09 2017.

[69] K. S. Sahoo and B. Sahoo, "CAMD: a switch migration based load balancing framework for software defined networks," *IET Networks,* vol. 8, no. 4, pp. 264-271, 2019/07 2019.

[70] N. T. Hai and D.-S. Kim, "Efficient load balancing for multi-controller in SDN-based mission-critical networks," presented at the 2016 IEEE 14th International Conference on Industrial Informatics (INDIN), 2016/07, 2016. Available: http://dx.doi.org/10.1109/indin.2016.7819196

[71] S. Yeo, Y. Naing, T. Kim, and S. Oh, "Achieving Balanced Load Distribution with Reinforcement Learning-Based Switch Migration in Distributed SDN Controllers," *Electronics,* vol. 10, no. 2, p. 162, 2021/01/13 2021.

[72] X. Ji, H. Yu, G. Fan, and W. Fu, "SDGR: An SDN-Based Geographic Routing Protocol for VANET," presented at the 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2016/12, 2016. Available: http://dx.doi.org/10.1109/ithings-greencom-cpscom-smartdata.2016.70

[73] D. C. Li, P.-H. Chen, and L.-D. Chou, "GAP4NSH: a genetic service function chaining with network service header for P4-based software-defined networks," *The Journal of Supercomputing,* vol. 79, no. 10, pp. 11495-11529, 2023/03/01 2023.

[74] N. M. Yungaicela-Naula, C. Vargas-Rosales, J. A. Pérez-Díaz, and M. Zareei, "Towards security automation in Software Defined Networks," *Computer Communications,* vol. 183, pp. 64-82, 2022/02 2022.

[75] J. Kim, Y. Kim, V. Yegneswaran, P. Porras, S. Shin, and T. Park, "Extended data plane architecture for in-network security services in software-defined networks," *Computers &amp; Security,* vol. 124, p. 102976, 2023/01 2023.

[76] X. Etxezarreta, I. Garitano, M. Iturbe, and U. Zurutuza, "Software-Defined Networking approaches for intrusion response in Industrial Control Systems: A survey," *International Journal of Critical Infrastructure Protection,* vol. 42, p. 100615, 2023/09 2023.

[77] M. Hassan, M. A. Gregory, and S. Li, "Multi-Domain Federation Utilizing Software Defined Networking—A Review," *IEEE Access,* vol. 11, pp. 19202-19227, 2023.

[78] S. Bhardwaj and A. Girdhar, "Network Traffic Analysis in Software-Defined Networking Using RYU Controller," *Wireless Personal Communications,* 2023/07/30 2023.

[79] H. Adhami, M. Alja'afreh, M. Hoda, J. Zhao, Y. Zhou, and A. El Saddik, "Suitability of SDN and MEC to facilitate digital twin communication over LTE-A," *Digital Communications and Networks,* 2023/06 2023.

[80] Y. Zhang and M. Chen, "Performance evaluation of Software-Defined Network (SDN) controllers using Dijkstra's algorithm," *Wireless Networks,* vol. 28, no. 8, pp. 3787-3800, 2022/08/20 2022.

[81] M. Rahouti, K. Xiong, Y. Xin, S. K. Jagatheesaperumal, M. Ayyash, and M. Shaheed, "SDN Security Review: Threat Taxonomy, Implications, and Open Challenges," *IEEE Access,* vol. 10, pp. 45820-45854, 2022.

[82] F. Bannour, S. Dumbrava, and D. Lu, "A Flexible GraphQL Northbound API for Intent-based SDN Applications," presented at the NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, 2022/04/25, 2022. Available: http://dx.doi.org/10.1109/noms54207.2022.9789785

[83] C. D. Bhowmik and T. Gayen, "Traffic aware dynamic load distribution in the Data Plane of SDN using Genetic Algorithm: A case study on NSF network," *Pervasive and Mobile Computing,* vol. 88, p. 101723, 2023/01 2023.

[84] A. Narwaria and A. P. Mazumdar, "Software-Defined Wireless Sensor Network: A Comprehensive Survey," *Journal of Network and Computer Applications,* vol. 215, p. 103636, 2023/06 2023.

[85] L. S. Peter, H. Kobo, and V. M. Srivastava, "A Comparative Review Analysis of OpenFlow and P4 Protocols Based on Software Defined Networks," in *Data Intelligence and Cognitive Informatics*, ed: Springer Nature Singapore, 2022, pp. 699-711.

[86] A. A. Abdul Ghaffar, A. Mahmoud, T. Sheltami, and M. Abu-Amara, "A Survey on Software-Defined Networking-Based 5G Mobile Core Architectures," *Arabian Journal for Science and Engineering,* vol. 48, no. 2, pp. 2313-2330, 2022/09/19 2022.

[87] O. J. Ibrahim and W. S. Bhaya, "Intrusion Detection System for Cloud Based Software-Defined Networks," *Journal of Physics: Conference Series,* vol. 1804, no. 1, p. 012007, 2021/02/01 2021.

[88] S. Sriram, K. Vishall, and T. R. Yaswanth, "Implementation of Load Balancing and Firewalls for Efficient Network Management in Software Defined Networks," presented at the 2022 International Conference on Edge Computing and Applications (ICECAA), 2022/10/13, 2022. Available: http://dx.doi.org/10.1109/icecaa55415.2022.9936257

[89] J. Miguel-Alonso, "A Research Review of OpenFlow for Datacenter Networking," *IEEE Access,* vol. 11, pp. 770-786, 2023.

[90] S. Ahmad, F. Jamil, A. Ali, E. Khan, M. Ibrahim, and T. Keun Whangbo, "Effectively Handling Network Congestion and Load Balancing in Software-Defined Networking," *Computers, Materials &amp; Continua,* vol. 70, no. 1, pp. 1363-1379, 2022.

[91] A. Fausto, G. Gaggero, F. Patrone, and M. Marchese, "Reduction of the Delays Within an Intrusion Detection System (IDS) Based on Software Defined Networking (SDN)," *IEEE Access,* vol. 10, pp. 109850-109862, 2022.

[92] M. Aldaoud, D. Al-Abri, A. Al Maashri, and F. Kausar, "Detecting and mitigating DHCP attacks in OpenFlow-based SDN networks: a comprehensive approach," *Journal of Computer Virology and Hacking Techniques,* 2023/02/21 2023.

[93] T. Yong Meng and R. Ayani, "Comparison of Load Balancing Strategies on Cluster-based Web Servers," *SIMULATION,* vol. 77, no. 5-6, pp. 185-195, 2001/11 2001.

[94] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Hanssgen, "A Survey of Payload-Based Traffic Classification Approaches," *IEEE Communications Surveys &amp; Tutorials,* vol. 16, no. 2, pp. 1135-1156, 2014.

[95] P. Rai and H. K. Deva Sarma, "A Survey on Application of LSTM as a Deep Learning Approach in Traffic Classification for SDN," in *Lecture Notes in Networks and Systems*, ed: Springer Nature Singapore, 2022, pp. 161-173.

[96] T. Linhares, A. Patel, A. L. Barros, and M. Fernandez, "SDNTruth: Innovative DDoS Detection Scheme for Software-Defined Networks (SDN)," ed: Research Square Platform LLC, 2022.

[97] N. Ahuja, G. Singal, D. Mukhopadhyay, and A. Nehra, "Ascertain the efficient machine learning approach to detect different ARP attacks," *Computers and Electrical Engineering,* vol. 99, p. 107757, 2022/04 2022.

[98] !!! INVALID CITATION !!! {}.

[99] T. Bilot, N. E. Madhoun, K. A. Agha, and A. Zouaoui, "Graph Neural Networks for Intrusion Detection: A Survey," *IEEE Access,* vol. 11, pp. 49114-49139, 2023.

[100] M. N. Munther, F. Hashim, N. A. Abdul Latiff, K. A. Alezabi, and J. T. Liew, "Scalable and secure SDN based ethernet architecture by suppressing broadcast traffic," *Egyptian Informatics Journal,* vol. 23, no. 1, pp. 113-126, 2022/03 2022.

[101] T. A. Alghamdi and N. Javaid, "A Survey of Preprocessing Methods Used for Analysis of Big Data Originated From Smart Grids," *IEEE Access,* vol. 10, pp. 29149-29171, 2022.

[102] S. Faezi and A. Shirmarz, "A Comprehensive Survey on Machine Learning using in Software Defined Networks (SDN)," *Human-Centric Intelligent Systems,* 2023/06/08 2023.

[103] A. Shirmarz and A. Ghaffari, "Performance issues and solutions in SDN-based data center: a survey," *The Journal of Supercomputing,* vol. 76, no. 10, pp. 7545-7593, 2020/01/30 2020.

[104] J. Wang and F. Biljecki, "Unsupervised machine learning in urban studies: A systematic review of applications," *Cities,* vol. 129, p. 103925, 2022/10 2022.

[105] J. Xie *et al.*, "A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges," *IEEE Communications Surveys &amp; Tutorials,* vol. 21, no. 1, pp. 393-430, 2019.

[106] P. Sepin, J. Kemnitz, S. R. Lakani, and D. Schall, "Comparison of Clustering Algorithms for Statistical Features of Vibration Data Sets," *arXiv preprint arXiv:2305.06753,* 2023.

[107] F. Deseure-Charron, S. Djebali, and G. Guérard, "Clustering Method for Touristic Photographic Spots Recommendation," in *Advanced Data Mining and Applications*, ed: Springer Nature Switzerland, 2022, pp. 223-237.

[108] P. Vial, M. Malagón, R. Segura, N. Palomeras, and M. Carreras, "GMM Registration: a Probabilistic scan matching approach for sonar-based AUV navigation," presented at the 2023 IEEE International Conference on Robotics and Automation (ICRA), 2023/05/29, 2023. Available: http://dx.doi.org/10.1109/icra48891.2023.10160697

[109] A. Ouadah, L. Zemmouchi-Ghomari, and N. Salhi, "Selecting an appropriate supervised machine learning algorithm for predictive maintenance," *The International Journal of Advanced Manufacturing Technology,* vol. 119, no. 7-8, pp. 4277-4301, 2022/01/11 2022.

[110] Á. Fernández, J. Bella, and J. R. Dorronsoro, "Supervised outlier detection for classification and regression," *Neurocomputing,* vol. 486, pp. 77-92, 2022/05 2022.

[111] S. Manickam *et al.*, "Labelled Dataset on Distributed Denial-of-Service (DDoS) Attacks Based on Internet Control Message Protocol Version 6 (ICMPv6)," *Wireless Communications and Mobile Computing,* vol. 2022, pp. 1-13, 2022/04/18 2022.

[112] Ö. Tonkal and H. Polat, "Traffic Classification and Comparative Analysis with Machine Learning Algorithms in Software Defined Networks," *Gazi Üniversitesi Fen Bilimleri Dergisi Part C: Tasarım ve Teknoloji,* vol. 9, no. 1, pp. 71-83, 2021/03/25 2021.

[113] M. M. Raikar, M. S M, M. M. Mulla, N. S. Shetti, and M. Karanandi, "Data Traffic Classification in Software Defined Networks (SDN) using supervised-learning," *Procedia Computer Science,* vol. 171, pp. 2750-2759, 2020.

[114] M. Perera Jayasuriya Kuranage, K. Piamrat, and S. Hamma, "Network traffic classification using machine learning for software defined networks," in *Machine Learning for Networking: Second IFIP TC 6 International Conference, MLN 2019, Paris, France, December 3–5, 2019, Revised Selected Papers 2*, 2020, pp. 28-39: Springer.

[115] J. S. Rojas, Á. R. Gallón, and J. C. Corrales, "Personalized Service Degradation Policies on OTT Applications Based on the Consumption Behavior of Users," in *Computational Science and Its Applications – ICCSA 2018*, ed: Springer International Publishing, 2018, pp. 543-557.

[116] S. Ismail and H. Reza, "Evaluation of Naïve Bayesian Algorithms for Cyber-Attacks Detection in Wireless Sensor Networks," presented at the 2022 IEEE World AI IoT Congress (AIIoT), 2022/06/06, 2022. Available: http://dx.doi.org/10.1109/aiiot54504.2022.9817298

[117] M. Bansal, A. Goyal, and A. Choudhary, "A comparative analysis of K-Nearest Neighbor, Genetic, Support Vector Machine, Decision Tree, and Long Short Term Memory algorithms in machine learning," *Decision Analytics Journal,* vol. 3, p. 100071, 2022/06 2022.

[118] L. Gao, D. Li, L. Yao, and Y. Gao, "Sensor drift fault diagnosis for chiller system using deep recurrent canonical correlation analysis and k-nearest neighbor classifier," *ISA Transactions,* vol. 122, pp. 232-246, 2022/03 2022.

[119] N. Jalal, A. Mehmood, G. S. Choi, and I. Ashraf, "A novel improved random forest for text classification using feature ranking and optimal number of trees," *Journal of King Saud University - Computer and Information Sciences,* vol. 34, no. 6, pp. 2733-2742, 2022/06 2022.

[120] M. Bahrami and M. Forouzanfar, "Sleep Apnea Detection From Single-Lead ECG: A Comprehensive Analysis of Machine Learning and Deep Learning Algorithms," *IEEE Transactions on Instrumentation and Measurement,* vol. 71, pp. 1-11, 2022.

[121] S. K. Singh, M. Kumar, and J. Singh, "Integration of Particle Swarm Optimization (PSO) and Machine Learning to Improve Classification Accuracy During Antenna Design," *Transactions on Electrical and Electronic Materials,* vol. 24, no. 3, pp. 258-266, 2023/05/11 2023.

[122] A. S. Alfoudi *et al.*, "Hyper clustering model for dynamic network intrusion detection," *IET Communications,* 2022/10/21 2022.

[123] N. H. Abed, M. J. Al-Dujaili, and S. A. Abbas, "Proposed an efficient multilevel dynamic bandwidth allocation (M-DBA) scheme for FiWi networks," *Optical and Quantum Electronics,* vol. 54, no. 10, 2022/08/21 2022.

[124] E. Reticcioli, G. D. Di Girolamo, F. Smarra, A. Torzi, F. Graziosi, and A. D'Innocenzo, "Modeling and Control of Priority Queueing in Software Defined Networks via Machine Learning," *IEEE Access,* vol. 10, pp. 91481-91496, 2022.

[125] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating system concepts essentials*. Wiley Publishing, 2013.

[126] A. S. Tanenbaum and H. Bos, "Modern operating systems. Pearson," ed: Pearson, 2015.

[127] K. Soleimanzadeh, M. Ahmadi, and M. Nassiri, "SD-WLB: An SDN-aided mechanism for web load balancing based on server statistics," *ETRI Journal,* vol. 41, no. 2, pp. 197-206, 2019.

[128] X. Sun and N. Ansari, "Latency Aware Workload Offloading in the Cloudlet Network," *IEEE Communications Letters,* vol. 21, no. 7, pp. 1481-1484, 2017/07 2017.

[129] V. Huang *et al.*, "Request Dispatching Over Distributed SDN Control Plane: A Multiagent Approach," *IEEE Transactions on Cybernetics,* pp. 1-14, 2023.

[130] G. Wang, Y. Zhao, J. Huang, and W. Wang, "The Controller Placement Problem in Software Defined Networking: A Survey," *IEEE Network,* vol. 31, no. 5, pp. 21-27, 2017.

[131] S. Kumari and S. K. Rath, "Performance comparison of SOAP and REST based Web Services for Enterprise Application Integration," presented at the 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2015/08, 2015. Available: http://dx.doi.org/10.1109/icacci.2015.7275851

[132] L.-D. Chou, Y.-T. Yang, Y.-M. Hong, J.-K. Hu, and B. Jean, "A Genetic-Based Load Balancing Algorithm in OpenFlow Network," in *Lecture Notes in Electrical Engineering*, ed: Springer Netherlands, 2013, pp. 411-417.

[133] E.-J. Wagenmakers and S. Brown, "On the linear relation between the mean and the standard deviation of a response time distribution," *Psychological Review,* vol. 114, no. 3, pp. 830-841, 2007.

[134] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the Social Network's (Datacenter) Network," presented at the Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, 2015/08/17, 2015. Available: http://dx.doi.org/10.1145/2785956.2787472

الملخص

عدم قدرة الشبكات التقليدية على توزيع الموارد بكفاءة، والتكيف مع أنماط المرور المتغيرة، والتعامل مع أنظمة الإدارة اللامركزية. تقترح الأطروحة استخدام أساليب حديثة بديلة، مثل الشبكات المعرفة بالبرمجيات (SDN)، للتعامل مع القيود الموجودة في الشبكات التقليدية. يؤدي فصل مهام التحكم في الشبكة وإعادة توجيه البيانات إلى إنشاء بنية شبكة مركزية وقابلة للبرمجة.

أحد التحديات المهمة في الشبكات التقليدية هو الطريقة المستخدمة لموازنة التحميل. عادةً ما تعتمد الشبكات التقليدية على خوادم مخصصة لأداء مهام موازنة التحميل. تتعامل هذه الخوادم مع المهمة المعقدة المتمثلة في توزيع حركة مرور الشبكة عبر موارد متعددة، وهذا الأسلوب له حدود من حيث المرونة وسرعة الحركة. في المقابل، تُحدث SDN ثورة في موازنة التحميل من خلال تمكين تنسيقها وإدارتها داخل وحدة تحكم SDN. يمكن برمجته لتوزيع حركة مرور الشبكة بذكاء بناءً على الظروف والسياسات الديناميكية.

اقترحت الأطروحة نهجًا جديدًا لموازنة التحميل في شبكات SDN من خلال اقتراح خوارزمية متكاملة ثلاثية النماذج، KNN-MLQLRL، والتي تجمع بين التعلم الآلي وقائمة الانتظار متعددة المستويات وجدولة موازنة التحميل (ML-MLQLBS). ويهدف هذا التكامل إلى إنشاء نموذج ذكي لموازنة الحمل الديناميكي، وبالتالي تحسين جودة الخدمة (QoS) للشبكة وتوزيع عبء العمل.

يتكون النموذج المقترح من سبع مراحل مختلفة. تشمل المرحلة الأولية إنشاء الحزم والتقاطها وتحليلها وتجميعها لإنشاء مجموعة بيانات، ومرحلة المعالجة المسبقة هي المرحلة الثانية. ثم يتم استخدام أساليب التجميع في المرحلة الثالثة لتعيين تسميات الفئة. تستخدم المرحلة الرابعة تصنيف التعلم الآلي (MLC)، وخوارزمية K-Nearest Neighbors (KNN) ، لتصنيف الأولويات والتنبؤ بها بشكل فعال. من ناحية أخرى، تستخدم المرحلة الخامسة طريقة قائمة الانتظار متعددة المستويات (MLQ) المقترحة للحصول على الأولويات من MLC وتخزينها قبل إرسالها إلى الخوادم. تم تطبيق طريقة جدولة موازنة التحميل (LBS) المقترحة في المرحلة السادسة. تستخدم هذه الطريقة خوارزمية جديدة تسمى تحميل الموارد الأقل (LRL) لتحديد الخادم الذي يحتوي على أقل وحدة معالجة مركزية وذاكرة. وأخيراً المرحلة السابعة وهي تصميم النموذج المقترح. يتضمن هذا النموذج نتائج المراحل السابقة وموازنة التحميل الديناميكية الذكية لتوزيع العمل بين الخوادم بشكل أفضل.

لمقارنة مدى جودة عمل النموذج المقترح مع طرق موازنة التحميل الستة الأخرى بناءً على أربعة من معلمات جودة الخدمة: وقت الاستجابة، وزمن الوصول، والإنتاجية، ودرجة موازنة التحميل. النموذج المقترح (KNN-MLQLRL) تحسن بشكل كبير في جميع المعلمات. حققت معلمة وقت الاستجابة في مجموعة البيانات 1 20%، وفي مجموعة البيانات 2، حققت 15%. كانت معلمة زمن الوصول 93% في مجموعة البيانات 1، وفي مجموعة البيانات 2، كانت 77%. حققت معلمة الإنتاجية في مجموعة البيانات 1 18%، وفي مجموعة البيانات 2، كان الأداء 7%. وأخيرًا، بلغت درجة معلمة موازنة التحميل في مجموعة البيانات 1 45%، وفي مجموعة البيانات 2، كانت 38%.

جمهورية العراق

وزارة التعليم العالي والبحث العلمي

جامعة بابل

كلية تكنولوجيا المعلومات

قسم شبكات المعلومات

موازنة التحميل الديناميكي الذكية لتحسين جودة الخدمة وتوزيع عبء العمل في الشبكات المعرفة بالبرمجيات

اطروحة مقدمة إلى

مجلس كلية تكنولوجيا المعلومات - جامعة بابل كجزء من متطلبات

نيل درجة الدكتوراه فلسفة في تكنولوجيا المعلومات - شبكات المعلومات

من قبل

مغرب عبد الرضا مكي عبد الرضا

بإشراف

أ.د. وسام سمير بهية

1445هـ **2023 م**