

**DESIGN FAULT TOLERANCE AND LOAD BALANCING  
MECHANISMS FOR MULTIPLE CONTROLLERS IN  
SOFTWARE DEFINED NETWORK**

Thesis Submitted for the Award of the Degree of

**DOCTOR OF PHILOSOPHY**

in

**Computer Applications**

By

**Deepjyot Kaur Ryait**

**Registration Number: 11816022**

Supervised By

**Dr. Manmohan Sharma (UID: 21909)**

**School of Computer Applications (Professor)**

**Lovely Professional University**



**L** OVELY  
**P** ROFESSIONAL  
**U** NIVERSITY

---

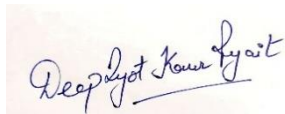
*Transforming Education Transforming India*

**LOVELY PROFESSIONAL UNIVERSITY, PUNJAB**

**2023**

## DECLARATION

I, hereby declared that the presented work in the thesis entitled “**Design Fault Tolerance and Load Balancing Mechanisms for Multiple Controllers in Software Defined Network**” in the fulfilment of the degree of **Doctor of Philosophy (Ph.D.)** is the outcome of research work carried out by me under the supervision of **Dr. Manmohan Sharma**, working as Professor, in the School of Computer Applications of Lovely Professional University, Punjab, India. In keeping with the general practice of reporting scientific observations, due acknowledgments have been made whenever the work described here has been based on the findings of other investigators. This work has not been submitted in part or full to any other University or Institute for the award of any degree.



**(Deepjyot Kaur Ryait)**

11816022


School of Computer Applications

Lovely Professional University,

Punjab, India

## CERTIFICATE

This is to certify that the work reported in the Ph.D. thesis entitled “**Design Fault Tolerance and Load Balancing Mechanisms for Multiple Controllers in Software Defined Network**” is submitted in fulfillment of the requirement for the reward of the degree of **Doctor of Philosophy (Ph.D.)** in the School of Computer Applications, is a research work carried out by **Deepjot Kaur Ryait**, Registration No. **11816022**, is a bonafide record of her original work carried out under my supervision and that no part of the thesis has been submitted for any other degree, diploma or equivalent course.



**(Dr. Manmohan Sharma)**

Professor

School of Computer Applications

Lovely Professional University, Punjab, India

**This work is dedicated to Almighty God,  
my respected parents, my sister, my brothers and my  
respected supervisor whose blessings, inspirations,  
continuous motivation and love has made it possible for  
me to undertake and complete my Ph.D. Thesis.**

# **ABSTRACT**

In the digital world, every person wants to be connected to know what is happening in the virtual world. Therefore, different users have different requirements based on their needs. Still, technologies are undergoing several changes; recently introduced a technology that attracts more attention in an academic and industrial area is the Software Defined Network (SDN); although SDN initially supports centralized logical control of the entire network; that offers facilities like programmability, adjustability, and dynamically reconfiguration of the networking elements as compared to the traditional networking. The framework of Software Defined Networks offers an abstraction network, which becomes too easy to achieve network reachability. This technology provides the provision of network services to upscale their infrastructure in a network with minimal disturbance. But in a traditional network, there is a strong interdependency between both functional components and the non-existence of programmability. For this reason, it is a complicated task to configure network devices as per the requirements of network users due to the predefined policies. Thus, it is not coping with modern enterprise user demand.

The main objectives of the thesis are to design a model for fault tolerance and load balancing in SDN. In a traditional network, it becomes more complex to manage the widespread adoption of a network because it is complicated to configure the network elements due to its predefined policies. If trying to reconfigure network elements, it becomes unmanageable sometimes. The main reason behind this is a vertical integration exists between both functional components of the traditional networking devices. So, it can be said that when adding a new feature, it always increases the cost or expensive and difficulty of configuring the elements due to the change in topology and functionality of the network. To overcome this situation, restructuring the infrastructure of the traditional networks as Software Defined Network (SDN). Decouple the control layer from the data layer in SDN; this improves network

performance. To overcome these challenges, the researchers are devoting more efforts toward the SDN controller to improve the performance of the networks in terms of reliability, scalability, high availability and resilience of services. Based on the study, the main objectives of the thesis would be to design a model for proactive fault tolerance in SDN to reduce single point failure, to design an adaptive algorithm for load balancing in SDN controllers, and validation of proposed fault tolerance and load balancing algorithm for SDN.

Software Defined Networks offer a novel model for networking that boosts network flexibility and programmability by dividing the control plane from the data plane. The controller receives the full control logic and offers a centralized, global logical representation of the network. The SDN controller has several responsibilities, like managing network behavior, dedicating and controlling the functionality of network elements, controlling the network management and complexity of the network, making a decision regarding regular network traffic in a network, etc. As a result, a controller is an essential part of SDN because it allows for the control of all network implementations and manipulations. However, this also raises the likelihood of a single controller becoming a single or central point of failure in the network. It causes the complete network to collapse as a result. As a consequence, a fault tolerance mechanism is needed that employs numerous controllers to decrease the network's single point of failure (SPOF).

The importance of having numerous controllers is that they improve the network's services' scalability, dependability, and high availability. During the simulation study, it is discussed what happens when a network contains a single point of failure and how to use various functions of many controllers to reduce the risk of a SPOF in the SDN network. The packet loss parameter gradually increases after a SPOF event happens in a network until the controller is once more in a working or operational state. Consequently, the network's numerous controllers can be used to solve this issue. In SDN, a centralized controller manages all responsibilities for the network. Then it becomes very difficult to accomplish their responsibilities simultaneously. In SDN networks, the use of many controllers results in a number of load balancing-related issues, including controller failure or cascading failure of

controllers. These issues include the controller becoming overwhelmed when the load surpasses its threshold value. Due to the direct effect on network performance, allocating the workload among the various SDN controllers is one of the more challenging tasks. Furthermore, the load balancing between the control planes and their scalability have a significant impact on the network's performance in the SDN environment. In order to efficiently utilize the network's overall view to manage congestion, an algorithm for load balancing across multiple SDN controllers is suggested for this purpose by using the queuing technique.

Through the separation of the control plane and data plane, Software Defined Network (SDN) offers an agile paradigm. Still, a solitary controller has a higher impact on failure in a network. However, having numerous controllers as the network's single point of failure can be avoided. Meanwhile, some challenges are counter like uneven traffic distribution between controllers which become an origin of cascaded failure of controllers; when a controller manages network traffic beyond its capacity. Then drop rate of packets is increasing exponentially in a network. This is happening due to the insufficiency of the load balancing technique. So, it is necessary to distribute the appropriate workload among the controllers. To propose a load balancing algorithm that gets rid of these challenges by evaluating an equilibrium state of distribution which describes the long-run probability of controllers by integrating Queuing Technique with Markov Continuous Chain, which aids to lessens the packet drop ratio in the network. The queuing method successfully manages network congestion for this purpose by taking a holistic view of the network. Along with the different queue items, it also offers the ability to calculate the packet delivery ratio and packet drop ratio. In order to manage packet loss rate in the network, it is necessary to ascertain the importance of queue size. These variables improve a network's quality of service (QoS).

The correlation and multiple regression model that can statistically explain the variation in the Packet Successfully Delivery in a network with the Queue Size, Total No. of Packets, and Packet Size in the network are then evaluated. The performance of the different SDN controller jobs is next compared to a suggested model or scheme in terms of round-trip time, average throughput, average bandwidth, ping

delay, etc. If the lessened value of round-trip time, the reliability of a network is increasing and the latency of the network is decreased. During simulation to analyze the proposed scheme, show more appropriate results as compared to other configurations of SDN controllers.



# **ACKNOWLEDGMENT**

I express my sincere thanks and deep gratitude to my guide and supervisor, Dr. Manmohan Sharma, Professor in the School of Computer Applications, Lovely Professional University, Punjab, for his unwavering support, suggestions, encouragement, guidance, and patience in helping me to accomplish this work. With great respect, I am expressing my gratitude to him for his valuable contribution to the progress and development of my research. I have benefited a lot because his help and guidance inspired me greatly. He possesses knowledge of immense breadth and depth, coupled with great talent and enthusiasm for research. He has led me into the area of networking (Software Defined Network) and simulated my research interest. He has taught me not only how to undertake research work. But also, how to communicate effectively in the form of a research paper. I am fortunate to be associated with him. His active participation, untiring efforts, affection, guidance, and approach have brought this work to the present stage.

I have received help and guidance from many other people in preparation for the thesis. But as a researcher, the guidance provided by my advisor and guide, Dr. Manmohan Sharma, is unparalleled. From framing the research problems to carrying out an analysis, from simulation to paper writing, and from the finer points of grammar to the finer points of teaching classes. He has spent an uncountable hour with me helping to complete this work. I am truly grateful to him for his kind help.

During this period, I may have been tired and disappointed many times, but I always believed and had the confidence that Dr. Manmohan Sharma is there to help and guide me, clear all my doubts, and provide me with strength and direction. He always found time whenever I needed for intelligent fruitful discussions. He always encouraged me whenever I became frustrated. I am also thankful to my supervisor for their valuable and timely suggestions and motivation. I feel very proud to have

worked with them. Without their inspiring enthusiasm and encouragement, this work could not be complete.

I am extremely grateful to Dr. Anil Sharma, Professor in the School of Computer Applications, Lovely Professional University, Punjab, who infused my passion for research and always encouraged me with parental guidance and moral support. I wish to express my gratitude to Lovely Professional University, Punjab, for providing an excellent environment for research and teaching facilities. I also extend my sincere thanks to the RDC (Research Degrees Evaluation Cell) and CRDP (Centre for Research Degree Programmes) of the Lovely Professional University, Punjab, for providing a proper conducive environment for the completion of this work.

I give my deepest thanks to my beloved parents, Papa Jagjit Singh Ryait and Mom Baljit Kaur Ryait; my sister, Dr. Jasmeet Kaur Ryait; and my brothers, Karanvir Singh Ryait and Gurdatar Singh Ryait, for their infinite love, inspiration, encouragement, and spirited support. I give thanks to God for the talent and abilities that enabled me to undertake this research.

I am particularly grateful to my supervisor for carefully reviewing the draft material of this thesis. Last but not least, I would like to take the opportunity to thank my mentors for their constant encouragement, hours of sitting together, and frequently lively discussions, which helped me and encouraged me to complete the work.

**Deepjyot Kaur Ryait**

School of Computer Applications  
Lovely Professional University,  
Punjab, India

Date: - 23<sup>th</sup> November, 2023

# **PREFACE**

Present and future generations of networks support different requirements for different users. Although, Software Defined Network initially started to support a centralized logical control of the entire network that offers facilities like programmability, adjustability, and dynamically reconfiguration of the networking elements to compare traditional networks. The framework of Software Defined Networks offers an abstraction network, which becomes too easy to achieve network reachability. This technology provides a centralized location for managing the provision of network services to upscale their infrastructure in a network with minimal disturbance. But in traditional networks, there is a strong interdependency between both functional components and the non-existence of the programmability. For this reason, it is a complicated task to configure network devices as per the requirements of network users due to the predefined policies. Thus, it is not coping with modern enterprise user demand.

In this thesis, they have presented that the SDN controller is a very crucial component of the network that manages all traffic on the network. Controller is responsible for taking routing decisions for the network, while forwarding devices are simply the forwarding elements in the data plane. The motive of the research work is to address fault tolerance and load balancing between multiple controller issues in Software Defined Network. Specifically, to evaluate an equilibrium state of distribution that describes the long-run probability of controllers by integrating Queuing Technique with the Markov Continuous Chain; which aids in lessening the packet drop ratio in the network. Our work is hereby presented in the form of a thesis containing seven chapters.

In Chapter 1, titled “Introduction to Software Defined Network,” has presents the evolution of the networking paradigm, describing the technical changes through various progressive generations of networks. The chapter presents the basic concepts

of Software Defined Network with its characteristics, history, essential elements of SDN, etc. The difference between both networking paradigms is represented in tabular form. The chapter consummates with concerning the current status of SDN networks.

In Chapter 2, titled "Architecture of Software Defined Network and OpenFlow Protocol," which describes the architecture of SDN and the operations of its planes. The next section of the chapter describes the OpenFlow protocol, flow table, and types of messages in OpenFlow. The different categories of implementation flow rules are discussed in the next section of the chapter.

Chapter 3 has been titled "Literature Review in Software Defined Network." In this chapter, has reviewed the literature on Traditional Networks, Software Defined Networks, and their protocols to bring out their merits under different conditions. Fault tolerance and load balancing in SDN are challenging tasks. Therefore, they have received a tremendous amount of attention from many researchers. Many researchers lighted on these issues in SDN as a further research direction.

Chapter 4, titled "Significance of Controller in Software Defined Networks," presents a controller acting as a critical and crucial component in SDN because it always provides a logical view of an entire network. If any fault occurs in the controller, then the operation of the entire network collapses. So, a fault tolerance mechanism is required to reduce a single point of failure in an SDN network by using multiple controllers.

In Chapter 5, titled " Load Balancing using Queuing Models in Multiple Controller Environment of SDN," an algorithm for load balancing in multiple controllers in SDN using the queuing technique is proposed, which effectively uses the global view of a network to control network congestion. To compare the M/M/1 Queue Model based on the likelihood of different system characteristics like  $L_s$ ,  $L_q$ ,  $W_s$ , and  $W_q$ , with infinite vs finite capacity. These variables aid in assessing the effectiveness of SDN controllers.

In Chapter 6, titled "Balancing through Probability Distribution in SDN," in this chapter, has proposed queuing technique with a Markov Continuous Chain that

effectively uses the global view of the network to control the congestion of a network. Moreover, to determine what is the significance of queue size for controlling packet drop rates in the network. These parameters enhance the quality-of-service (QoS) of a network. The performance is evaluated by the correlation and the multiple regression model can statistically explain the variation in Packet Successfully Delivery in the network with the Queue Size, Total No. of Packets, and Packet Size in the network. Then it compares the performance of the various roles of SDN controllers with a proposed model or scheme in terms of parameters such as average throughput, average bandwidth, ping delay, and so on.

Chapter 7 is titled "Conclusion and Future Work," which concludes the thesis by discussing the observations and findings acquired from the proposed method/technique in the thesis and discusses the research work that can be carried out in the future as an extension of the present work. To conclude, they believe their research can help other researchers identify challenges and new research directions in the area of fault tolerance and load balancing in the Software Defined Network. The following chapter describes the additional work that can be done using the research offered in this thesis.

# TABLE OF CONTENTS

## Contents

<b>DECLARATION .....</b>	<b>ii</b>
<b>CERTIFICATE.....</b>	<b>iii</b>
<b>ABSTRACT .....</b>	<b>v</b>
<b>ACKNOWLEDGMENT.....</b>	<b>ix</b>
<b>PREFACE .....</b>	<b>xi</b>
<b>LIST OF ABBREVIATIONS .....</b>	<b>xix</b>
<b>LIST OF SYMBOLS .....</b>	<b>xxi</b>
<b>LIST OF TABLES .....</b>	<b>xxii</b>
<b>LIST OF FIGURES .....</b>	<b>xxiv</b>
<b>Chapter 1 Introduction to Software Defined Network</b>	<b>1-12</b>
1.1 Introduction.....	1
1.2 Definition of Software Defined Network .....	2
1.3 Characteristics of SDN .....	2
1.4 Traditional Network v/s Software Defined Network .....	3
1.5 History of SDN .....	7
1.6 Essential Elements of SDN.....	8
1.6.1 Forwarding Devices.....	8
1.6.2 Data Plane .....	8
1.6.3 Southbound Interface (SBI) .....	9
1.6.4 Control Plane.....	9

1.6.5 Northbound Interface (NBI).....	9
1.6.6 Eastbound/Westbound Interfaces .....	9
1.6.7 Application Plane .....	9
1.7 Origin of Software Defined Network.....	10
1.8 Basic Concepts of SDN.....	10
1.9 Components/Attributes of SDN.....	10
1.10 Current Status of SDN.....	11
<b>Chapter 2 Architecture of Software Defined Network and OpenFlow Protocol</b>	<b>14-28</b>
2.1 Architecture of Software Defined Network.....	14
2.2 Data Plane.....	16
2.3 Control Plane .....	17
2.4 Application Plane .....	17
2.5 OpenFlow Protocol in SDN.....	19
2.6 Flow Table.....	19
2.6.1 Type of Message .....	21
2.7 Implementation of Flow Rules.....	23
2.7.1 Reactive Method.....	23
2.7.2 Proactive Method.....	24
2.8 OpenFlow Pipeline and Group Table.....	25
2.9 Summary.....	28
<b>Chapter 3 Literature Review in Software Defined Network</b>	<b>30-54</b>
3.1 Introduction.....	30
3.2 Traditional Networks.....	31
3.3 Software Defined Networks.....	32

3.3.1 SDN Protocols.....	49
3.3.2 OpenFlow Protocol.....	49
3.4 Observations and Research Gaps.....	50
3.5 Objectives .....	52
3.6 Summary.....	53
<b>Chapter 4 Significance of Controller in Software Defined Networks</b>	<b>56-82</b>
4.1 Introduction.....	56
4.2 Related Work .....	57
4.3 Problem Formulation.....	58
4.4 Why Controller is a Crucial Component in SDN .....	60
4.5 Single Controller v/s Multiple Controllers .....	61
4.6 Type of Multiple Controller's in SDN .....	62
4.7 Simulation and Result.....	66
4.7.1 Mininet Simulator.....	67
4.8 Multi-deployment of Controllers in SDN.....	75
4.9 Conclusion .....	82
<b>Chapter 5 Load Balancing using Queuing Models in Multiple Controller Environment of SDN</b>	<b>84-112</b>
5.1 Introduction.....	84
5.2 Related Work .....	85
5.3 Problem Formulation.....	86
5.3.1 Need of Load Balancing in SDN Controllers .....	86
5.4 Proposed Design of Algorithm for Load Balancing.....	87
5.5 Queuing Model M/M/1: $\infty/\infty$ versus M/M/1: $N/\infty$ and its Simulation.....	90
5.5.1 NS2 Simulator .....	97



5.5.2 Numerical Evaluation and Result.....	108
5.6 Conclusion .....	111
<b>Chapter 6 Balancing through Probability Distribution in SDN</b>	<b>114-154</b>
6.1 Introduction.....	114
6.2 Related Work .....	115
6.3 Problem Formulation.....	116
6.4 Proposed Approach .....	117
6.4.1 Markov Chain.....	119
6.4.2 Transition Probability of ‘n’ Steps .....	119
6.4.3 Equilibrium State of Probability.....	121
6.4.4 Equilibrium State in a Queue .....	122
6.4.5 Pseudo Code for Load Balancing in SDN Controllers by using the Queuing Technique with the Markov Chain .....	123
6.5 Simulation and Evaluation of Result.....	125
6.5.1 Queue Model.....	125
6.5.2 Various Queue Objects .....	127
6.5.3 Equilibrium State of Controllers .....	129
6.5.4 Significance of Queue Size .....	137
6.5.5 Correlation Matrix .....	140
6.5.6 Multiple Regression Model.....	140
6.5.7 Performance Evaluation of Controllers .....	143
6.6 Conclusion .....	153
<b>Chapter 7 Conclusion and Future Work</b>	<b>156-158</b>
7.1 Conclusion .....	156
7.2 Future Work.....	157

<b>REFERENCES</b>	<b>160-175</b>
<b>LIST OF PUBLICATIONS</b>	<b>177-178</b>
<b>LIST OF CONFERENCES</b>	<b>180-181</b>

# **LIST OF ABBREVIATIONS**

4D	Decision, Dissemination, Discovery and Data Plane
API	Application Programming Interface
BGP	Border Gateway Protocol
CAPEX	Capital Expenditure
CDN	Content Delivery Network
CLI	Command-Line Interface
DCN	Data Center Networks
EID	End Point Identifiers
FIB	Forwarding Information Base
IETF	Internet Engineering Task Force
IP Address	Internet Protocol Address
JSON	JavaScript Object Notation
LISP	Locator/Identifier Separation Protocol
LLDP	Link Layer Discovery Protocol
MAC Address	Media Access Control Address
NAM	Network Animator
NBI	Northbound Interface
NetConf	Network Configuration Protocol
NFV	Network Function Virtualization
NOS	Network Operating System
O&M	Operational and Management

OF	OpenFlow
ONF	Open Networking Foundation
ONS	Open Networking Summit
OPEX	Operational Expenditure
OVSDB	Open vSwitch Database
QoS	Quality of Services
REST	REpresentational State Transfer
RIB	Routing Information Base
RLOC	Route Locators
RPC	Remote Procedure Call
RTT	Round-trip Time
SBI	Southbound Interface
SDN	Software Defined Network
SD-VPN	Software Defined Virtual Private Network
SPOF	Single Point of Failure
SSL	Secure Socket Layer
TCAM	Ternary Content Addressable Memory
TCL	Tool Command Language
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
XML	Extensible Markup Language

# LIST OF SYMBOLS

$\lambda$	Arrival Rate of a Packet
$\mu$	Service Rate of a Packet
$\rho$	Traffic Intensity or Utilization Factor
$P_0$	Idle Time
$L_s$	Length of a System
$L_q$	Length of a Queue
$W_s$	Waiting Time of a System
$W_q$	Waiting Time of a Queue
$P[X_{t+1}]$	Probability of the Future State
$P[X_t]$	Probability of the Present State
$p_{ij}$	Conditional Probability of the State
$\pi_0$	Initial Probability of a State
$\pi(k)$	Vector State of Probabilities for period k
$P$	Transition Probability of Matrix
$y$	Response Variable
$\beta_0$	Slope Intercept Coefficient
$X_1 + \dots + X_k$	Explanatory Variables
$\beta_1 + \dots + \beta_k$	Coefficient of Variables

# **LIST OF TABLES**

Table 1:	Traditional Networks versus Software Defined Networks	5
Table 2:	Difference between Reactive and Proactive Method	24
Table 3:	List of Action Set in OpenFlow	25
Table 4:	Various value of Group Type in OpenFlow	27
Table 5:	Literature Review on SDN Controller's	48
Table 6:	Simulation Parameters of Setup	68
Table 7:	Compare the Result of Controllers in SDN	74
Table 8:	Performance Metrics of Controllers w.r.t. Round-trip Time	78
Table 9:	Comparison of Controllers w.r.t. Average Throughput, Average Bandwidth and Ping Delay	81
Table 10:	Compare various Parameters in both Models	96
Table 11:	Simulation Parameters	98
Table 12:	Arrival and Service Rate of the Controllers in the Network	108
Table 13:	Comparison of Parameters of Controllers	110
Table 14:	Abbreviation of Variables used in Algorithm	123
Table 15:	Equilibrium State for Two Controllers	130
Table 16:	Equilibrium State of Two Controllers in Different Cases	134
Table 17:	Equilibrium State for Three Controllers	135
Table 18:	Equilibrium State for Four Controllers	136
Table 19:	Effect of Queue Size w.r.t. other Parameters	138
Table 20:	Performance Comparison of Controllers in Round-trip Time Metric	144

Table 21: Performance Evaluation of Controllers with the Proposed Model	147
Table 22: Performance Evaluation of Controllers with the Proposed Model w.r.t. Average Throughput, Average Bandwidth and Ping Delay	150
Table 23: Quantitative Analysis of Parameters	153

# **LIST OF FIGURES**

1.1	Traditional Networks	6
1.2	Software Defined Networks	7
2.1	Layer Structure of Software Defined Network	15
2.2	Architecture of Software Defined Network	16
2.3	SDN Paradigm work on these Four Pillars	18
2.4	OpenFlow Protocol Architecture	20
2.5	Format of the Flow Entries	20
2.6 (a)	Controller-to-Switch Messages	22
2.6 (b)	Asynchronous Messages	22
2.6 (c)	Synchronous Messages	22
2.7	Packet Forwarding Process in Reactive Method	23
2.8	Notify to the Controller by using PACKET-IN Message	24
2.9	Packet Flow Process in the Pipeline of OpenFlow	26
2.10	Format of Group Table Entries	26
2.11	Format of Meter Table in OpenFlow 1.3	27
4.1	Use of LLDP Packet between the Controller and Switch in SDN	61
4.2	Single Controller v/s Multiple Controllers in SDN	62
4.3	Various Role of the Multiple Controller in SDN	63
4.4	Behavior of Multiple Controller in SDN	64
4.5	Pseudo Code to Reduce Single Point of Failure in SDN	66
4.6 (a)	Run a SDN Controller in a Terminal	69



4.6 (b)	Run a topology in a Terminal through Mininet	69
4.6 (c)	Host Unreachable after killing the Controller	70
4.6 (d)	Simulation in a Single Controller in SDN	70
4.7 (a)	Run an Equal Controller in a Terminal at Port Number 6653	71
4.7 (b)	Run an Equal Controller in a Terminal at Port Number 6654	71
4.7 (c)	Duplicate Packets generated during Simulation of Equal Controller	71
4.7 (d)	Simulation in Equal Controller in SDN	72
4.8 (a)	Run the Master Controller in a Terminal at Port Number 6653	72
4.8 (b)	Run the Slave Controller in a Terminal at Port Number 6654	73
4.8 (c)	Run a topology by using Python API in the Mininet	73
4.8 (d)	No Duplicate Packets are generated in Master-Slave Simulation	73
4.8 (e)	Simulation in Master-Slave Controller in SDN	74
4.9 (a)	Overall CPU Utilization in Equal Controller	76
4.9 (b)	Overall CPU Utilization in Master-Slave Controller	76
4.10 (a)	Overall Memory Utilization in Equal Controller	77
4.10 (b)	Overall Memory Utilization in Master-Slave Controller	77
4.11	Comparison of CPU Utilization in Multiple Controller	78
4.12 (a)	Performance Metrics of Controllers in term Minimum RTT	79
4.12 (b)	Performance Metrics of Controllers in term Average RTT	79
4.12 (c)	Performance Metrics of Controllers in term Maximum RTT	80
4.12 (d)	Performance Metrics of Controllers in term Mean Deviation RTT	80
4.13 (a)	Comparison between Controllers w.r.t. Average Throughput, Average Bandwidth and Time Duration	81
4.13 (b)	Performance of Controllers w.r.t. Ping Delay	82
5.1	Use of Queuing concept in SDN Controller	88

5.2	Algorithm for Load Balancing in Multiple Controllers	89
5.3	Flowchart for Load Balancing in SDN Controllers	90
5.4 (a)	Probability of events occur in time interval $(t+\delta t)$	91
5.4 (b)	Possible of events occur in time interval $(t+\delta t)$	92
5.4 (c)	Representation of Queue Model $M/M/1: \infty/\infty$ in Network	94
5.4 (d)	Representation of Queue Model $M/M/1: N/\infty$ in Network	95
5.5	Compare various parameters $L_s, L_q, W_s$ & $W_q$ in both Models	96
5.6 (a)	$M/M/1$ Queue Model with Infinite Capacity	99
5.6 (b)	$M/M/1$ Queue Model with Finite Capacity	99
5.6 (c)	Instantaneous Throughput in Queue Model with Infinite Capacity	100
5.6 (d)	Instantaneous Throughput in Queue Model with Finite Capacity	100
5.6 (e)	Instantaneous Goodput in Queue Model with Infinite Capacity	101
5.6 (f)	Instantaneous Goodput in Queue Model with Finite Capacity	101
5.6 (g)	Instantaneous Delay in Queue Model with Infinite Capacity	102
5.6 (h)	Instantaneous Delay in Queue Model with Finite Capacity	102
5.6 (i)	Comparison between both Queue Models w.r.t. Queue-Size	103
5.6 (j)	Comparison between both Queue Models w.r.t. Arrived Packets	103
5.6 (k)	Comparison between both Queue Models w.r.t. Departed Packets	104
5.6 (l)	Comparison between both Queue Models w.r.t. Dropped Packets	104
5.6 (m)	Comparison between both Queue Models	105
5.7	Comparison between both Queue Model in Average Throughput, Average Delay and Packet Delivery Ratio	105
5.8 (a)	Flowchart for Load Balancing in SDN Controllers with queue monitoring variable	106
5.8 (b)	Algorithm for Load Balancing in Multiple Controllers	107

5.9	Scenario of Network using M/M/1: N/∞ Model for Load Balancing in Software Defined Network	108
5.10	Probability of Traffic Intensity of the Controllers	109
5.11	Probability w.r.t. $P_0$ and No Queue occur in the System	109
5.12	Probability w.r.t. $L_s$ , $L_q$ , $W_s$ & $W_q$ occur in the System	111
6.1	Use of Queuing Technique in SDN Controller	119
6.2	Transition Probability of 'n' step time period	120
6.3	Equilibrium State in Queue	122
6.4	Pseudo Code for Load Balancing in SDN Controllers	125
6.5 (a)	Comparison between both Queue Models	126
6.5 (b)	Comparison between both Queue Model in Average Throughput, Average Delay and Packet Delivery Ratio	126
6.6 (a)	Number of Packets Drop in various Queue Object	127
6.6 (b)	Probability of Packet Delivery Ratio in various Queue Object	128
6.6 (c)	Probability of Packet Drop Ratio in various Queue Object	128
6.7	Graphically Representation of Controllers	129
6.8	Equilibrium State of Two Controllers	131
6.9	A Few Cases of Equilibrium State of Two Controllers	133
6.10	Graphically Representation of an Equilibrium State of Controllers in Different Cases	133
6.11	Equilibrium State of Three Controllers	136
6.12	Equilibrium State of Four Controllers	137
6.13	Drop Rate of Packets v/s Queue Size	138
6.14	Packet Delivery Ratio v/s Queue Size	139
6.15	Packet Drop Ratio v/s Queue Size	139

6.16	Correlation Matrix between various Parameters	140
6.17	Result of Multiple Regression Model of Three Variables	141
6.18	Result of Multiple Regression Model of Two Variables	142
6.19	Classification of Controllers in Software Defined Network	144
6.20	Comparison between Controllers in term Minimum RTT	145
6.21	Comparison between Controllers in term Average RTT	145
6.22	Comparison between Controllers in term Maximum RTT	146
6.23	Comparison between Controllers in term Mean Deviation RTT	146
6.24	Performance Evaluation w.r.t. Min Delay, Max Delay, Avg Delay, Avg Jitter and Delay Standard Deviation	148
6.25	Performance Evaluation w.r.t. Bytes Received	148
6.26	Performance Evaluation w.r.t. Average Bitrate	149
6.27	Performance Evaluation w.r.t. Average Packet Rate	149
6.28	Performance Evaluation of Controllers w.r.t. Throughput by Gnuplot	151
6.29	Performance Evaluation of Controllers w.r.t. Bandwidth by Gnuplot	151
6.30	Performance Evaluation of Controllers with the Proposed Model	152
6.31	Performance Evaluation of Controllers w.r.t. Ping Delay	152

# **Chapter 1**

## **Introduction to Software Defined Network**

# **1 CHAPTER**

## **Introduction to Software Defined Network**

### **1.1 Introduction**

Today, Network Communication has a great impact on the life of human beings due to the growth or evolution of the internet. Communication is a way to share and exchange views, information, thoughts, emotions, and knowledge with another. So, an efficient communication system is required to accomplish the needs of a user. At present, the internet influences the life of a human being. It is difficult to manage daily routines when try to imagine living without the internet. Therefore, the internet offers a way to establish a digital society where everyone wants to interact and stay updated on what is occurring online. Still, technology is undergoing several changes; a recently introduced technology that attracts more attention in academic and industrial areas is the Software Defined Network (SDN). SDN aims to create an environment of network with efficiency, scalability, adaptability, flexibility, reliability, etc.

Traditionally, the network was more complex and harder to manage in its widespread adoption. Software Defined Networks (SDN), an essential technology with many potential applications in the sector, are created in order to address this issue by redesigning the current/traditional network infrastructure [6].

Today, every commercial enterprise is willing to earn maximum profit in the network field by offering various communication facilities. Any failure that occurs during the communication of the network results in a heavy loss of revenue. It is

essential to overcome this situation by increasing the availability of networks and minimizing revenue loss [16].

SDN greatly boosts network efficiency. Network management is made simpler by software defined networking. SDN provides several facilities like programmability, adjustability, and dynamically reconfiguration of the networking elements. Currently, Software Defined Networking paradigm is supported by large industries such as Microsoft, Google, Amazon, Cisco, Juniper, Facebook, HP, IBM, Samsung, etc.

## **1.2 Software Defined Network**

"By decoupling/disassociating the control plane from the data plane in a network," is how the term "Software Defined Network" (SDN) is described. SDN is an upcoming network architecture that provides programmability, simplicity of management and adaptability, dynamic configuration of network components, control, and efficient optimisation of network resources, by splitting the network management plane and forwarding plane. SDN makes it possible for networks to be directly connected to apps via application programming interfaces. (APIs).

SDN is an architecture that separates the control plane from the data forwarding function in networking devices. SDN also offers centralised control management, and enhances network flexibility and management, making it possible to design a dynamic, adaptable network architecture.

## **1.3 Characteristics of SDN**

The architecture of a Software Defined Network defines a novel way of a networking system that can be built by using a combination of hardware network commodities with software-based technologies and openness. Additionally, SDN offers a centralised network structure that can interact with the rest of the network. The framework of Software Defined Networks offers an abstraction network, which

becomes too easy to achieve network reachability. The various characteristics of Software Defined Networks are listed below:

1. Reduce the complexity of a network by decoupling the control plane from the data plane.
2. SDN controller provides a logically centralized view of a network for maintaining the network intelligence.
3. Using APIs, users can easily deploy applications and services.
4. Network control is directly programmable due to the separation of functional components.
5. Substantially reduce manual configuration in the network.
6. A readily programmable network is eliminated manual configuration.
7. SDN is an agile model which abstracts the control from the forwarding function to adjust dynamic changes.
8. Due to centralized control, forwarding elements can be configured at scale.
9. Controller interacts with network elements through APIs.
10. Easy to achieve network reachability through SDN as compared to the traditional networks.
11. Vendor Neutrality is achieved in SDN networks because instructions are provided by SDN Controller.
12. Open Standard-based is attained/accomplished in SDN by supporting standard protocols for communication between devices from multiple vendors and maintaining a common software environment.

## **1.4 Traditional Network v/s Software Defined Network**

Software Defined Networking is the most popular way to deploy applications in a network by organizations with a faster rate and decrease the overall cost of deployment. The framework of Software Defined Networks offers an abstraction network, which becomes too easy to achieve network reachability. This technology provides a centralized location for managing the provision of network services. Software Defined Network offers an option to upscale their infrastructure in a network with minimal disturbance/disruption. But in a traditional network, there is



a strong interdependency between both functional components and the non-existence of programmability. For this reason, it is a complicated task to configure network devices as per the requirements of network users due to the predefined policies. As a result, it is unable to meet the needs of modern enterprise users.

When reconfiguring these traditional network devices is tried results in a fault, imbalance, or alter the configuration of these devices within surges the complexity of the network [6,16]. Due to the traditional network's lack of programmability, it is difficult and expensive to add new functionality. This is because changing a network's topology is unprofitable because it raises the network's capital and operating costs. Additionally, if any changes are attempted in the control plane, all network devices must have new firmware installed on them or have their hardware upgraded.

A traditional network primarily relies on physical infrastructure, such as switches and routers, to build a communication connection between them, whereas a software defined network is built on a software-based network rather than hardware. SDN, on the other hand, is built on software that enables users to manage resources via a control plane as opposed to engaging with actual infrastructure.

The reason behind this rigidity of traditional networks is the vertical integration of both functional components which are bonded together inside the forwarding elements [6,16]. While the past decade, the failure of a link increased the inflexibility in the network due to the loss of perceptibility of operators over their network. The reason for that switch behavior is like a black box that is written by multiple vendors. It is to prevent the network operators to modify their implementation to satisfy the requirements of their customers. Thus, traditional networks arose many difficulties in handling the data transmission in the network. The Software Defined Network (SDN), which improves the network's efficiency by differentiating the control plane from the data plane, therefore is necessary to restructure a traditional network.

A software defined network has an isolated control plane and data plane. As a result, it provides programmability for network components, which lowers the network's complexity. It becomes simple and easy to modify or adjust network policies because the SDN controller provides a logically centralised picture of the complete

network. Additionally, SDN offers features like flexibility, centralised view control, reduced complexity, and lower network system costs. Thus, SDN's position as networking's future is provided through network innovation. More flexibility is available than in the traditional network thanks to the cutting-edge SDN paradigm. It enables users the ability to dynamically add additional network functionality in the shape of applications.

Software Defined Networking is a modern paradigm for networking, which offers increased programmability, more adaptability, and enhanced flexibility along with easy manageability, adjustability, and dynamical reconfiguration of network elements when compared with the traditional network paradigm. Therefore, it's crucial to redesign the conventional network (Software Defined Network) in order to increase network efficiency by splitting the management or control plane from the data plane. It provides two main benefits such as:

- 1) Completely control logic of the network is transferred to the controller.
- 2) Network devices function in the data layer as a straightforward forwarding element.

As a result, Software Defined Networking offers a cutting-edge networking paradigm that can assist in quickly satisfying customer needs. Moreover, SDN improves network control which permits the network provider to rejoiner the changing business requirements [1-9,14-17]. So, several industries are supporting the SDN paradigm like Microsoft, Google, VMware, IBM, Cisco, Juniper, etc. The differences between the two networks are depicted diagrammatically in Figures 1.1 and 1.2 as well as in Table 1.

Table 1: Traditional Networks versus Software Defined Networks

<b>Traditional Networks</b>	<b>Software Defined Networks (SDN)</b>
Hardware-based Network.	Software-based Network.
A strong bond exists between both functional components.	In SDN, a strong bond does not exist between both functional components.
Traditional Networks maintain a routing table in every switch.	SDN maintains a flow table for every switch.

Lack of programmability in a network.	Increase programmability of network.
Do not provide centralized control of the network.	The controller provides centralized control of an entire network.
Rigidly of adaptability in the network due to a tightly coupled bond exist in functional components.	More adaptability offers in the network due to a centralized control.
Do not provide network virtualization in traditional networks.	Network Virtualization provides in SDN.
Do not provide vendor neutrality. It works in the vendor-specification environment.	It provided vendor neutrality.
Little flexibility is provided as compared to SDN networks.	More flexible than traditional networks due to programmability.
The maintenance cost is higher.	The maintenance cost is less.
The operational and capital cost is high.	The operational and capital cost is less.

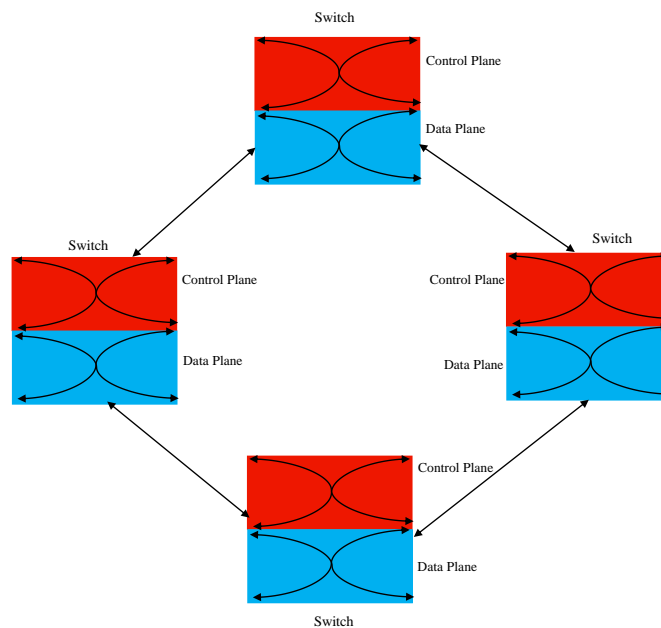


Figure 1.1: Traditional Networks

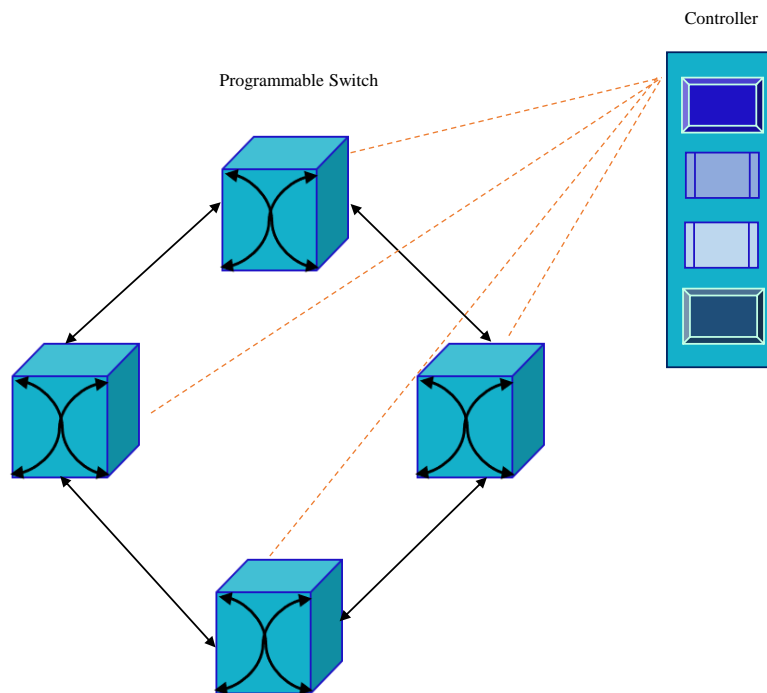


Figure 1.2: Software Defined Networks

## 1.5 History of SDN

Early in the year 2000, there was a sudden increase in demand for networking services with greater flexibility and reliability due to the internet. Many initiatives are currently being made to accomplish innovation in the networking industry by splitting the control layer from the data layer. The network architecture was changed by the 4D initiative to include the four planes of decision, dissemination, discovery, and data. Then, utilising two components, Ethan (an extension of SANE) addresses some 4D features: (1) a centralised controller has maintained a network-wide security policy; and (2) Ethan switches have received the forwarding rules from a controller. The two primary flaws of Ethan are the absence of knowledge about network nodes and users, and the requirement for control over flow level routing therefore Ethan was not adopted as a result [16].

Martin Casado et al. [25, 26] proposed a clean-slate security architecture (SANE) to control security policies in a centralized manner instead of doing it at the edge as normally done in 2006. Security is a fundamental goal to redesign an architecture.

Its main goal is to establish an architecture that supports simplicity, implements security at the link layer, and hides all topology and service information from unauthorized parties. The OpenFlow project originated the idea of a Software Defined Network. Then Stanford publishes the OpenFlow V1.0.0 specs. In 2009 Martin Casado co-founds Nicira (Nicira network founded). The Open Networking Foundation (ONF) is formed and the first open networking summit in 2011. Then many industries like Juniper, and Cisco are incorporated with SDN. The SANE and its replacement Ethan laid the foundation for the start of an OpenFlow. The OpenFlow (OF) protocol, a brand-new technology that is implemented in a university campus' network, was created by a study team at Stanford University. One of the most popular southbound interfaces in the SDN is the OpenFlow protocol, which facilitates interaction between both the data and control planes [6,8,16,27].

## **1.6 Essential Elements of SDN**

The Software Defined Network offers a framework of abstraction in a network that increases the network reachability; it is too easy to achieve in SDN compared to traditional networks. Identify different elements that are considered essential in Software Defined Networking explained below [6,15,16,64,66]:

### **1.6.1 Forwarding Devices:**

Devices in the data plane that carry out different network operations, like packet forwarding, can be either hardware or software-based. These forwarding components contain clearly specified instruction sets or flow rules that enable them to respond to incoming network packets by, for example, passing them to a controller or dropping them. The SDN controller installs these instructions over the forwarding elements' southbound interfaces.

### **1.6.2 Data Plane:**

The lowest layer of the software defined network paradigm is the data plane; basically, it is a collection of forwarding elements that are interconnected to each

other in the plane. They act as a simple forwarding element in the network whose behavior is controlled by a controller.

### **1.6.3 Southbound Interface (SBI):**

A southbound interface is existing between the control plane and forwarding plane whose responsibility is to provide communication between both planes. It also calls by the name southbound APIs. The flow of southbound interfaces can be going downward in SDN architecture.

### **1.6.4 Control Plane:**

A key component of the Software Defined Networking architecture is the control plane. It's commonly referred to as a controller. The control plane programmes every forwarding device using the southbound API because it offers a centralised logical view of the complete network. An SDN controller controls the flows of the switches and routers in the data plane through the southbound interfaces while managing the network applications and business logic via the northbound interfaces to implement network intelligence.

### **1.6.5 Northbound Interface (NBI):**

Communication among the application plane and the control plane is made possible by a northbound interface. Northbound APIs is another name for it. The flow of northbound interfaces can be going upward in the SDN paradigm.

### **1.6.6 Eastbound/Westbound Interfaces:**

This interface is used when multiple controllers are communicated in the network; because ensuring general compatibility and interoperability between multiple controllers is crucial.

### **1.6.7 Application Plane:**

The application plane is a top layer of SDN that includes a set of network applications or network services like routing, load balancing, monitoring, and so on. Through this plane, network users can control and manage the network. Because these applications define policies that are implemented by a controller via southbound APIs in forwarding devices.

## **1.7 Origin of Software Defined Network**

A pioneer of Software Defined Networks was Martin Casado. He is a co-founder of the Nicira networks as well as a software developer, business owner, and investor. In 2006, Martin Casado, a Ph.D. student at Stanford University, and his team proposed the clean-slate security architecture (SANE) to control security policies in a centralized manner. No new feature was available on the switch until the hardware was upgraded. Their work was motivated by the fact that software is separated and upgraded independently. In 2008, the idea or concept of a Software Defined Network was initiated or originated from the OpenFlow project, and Stanford published OpenFlow V1.0.0 specs in 2009. The Open Networking Foundation (ONF) was formed and the first Open Networking Summit (ONS) in 2011. Many companies like Juniper and Cisco announced incorporation with SDN.

## **1.8 Basic Concepts of SDN**

The Software Defined Network defines an innovative design way for a networking system that can be constructed by a combination of hardware network commodities with software-based technologies and openness. Furthermore, SDN provides a centralized structure or view of a network that can communicate with the rest of the network. The basic concepts of Software Defined Networks are listed below:

- To separate switch hardware from control logic.
- To centralized the development of a control logic or mechanism.
- Application Programming Interfaces allow for communication between the different planes (application plane, control plane, and data plane).

## **1.9 Components/Attributes of SDN**

SDN pledges to increase a network's flexibility, agility, and manageability by centralising control administration and abstracting the control plane from the job of

data forwarding in networking elements. The main components/attributes required for the Software Defined Networks are given below:

- Hardware Switches
- Controller
- Applications
- Flow Rules
- Application Programming Interfaces (APIs).

## **1.10 Current Status of SDN**

Software Defined Networking provides an innovative paradigm of networking that can help to fulfill the requirement of a user on demand. Moreover, SDN improves network control which permits the network provider to respond to the changing business requirements quickly. So, several industries are supporting the SDN paradigm. A company such as the Google has started to implement SDN in their data center networks. It is required to transform the current network with SDN in a phased manner. The operational costs and delays that occur due to a link failure can be significantly minimized in SDN. The recent deployment at Tribune Media, which used VMware NSX to transfer more than 140 apps to the company's new SDN infrastructure, serves as an illustration of an SDN in action. By dispersing network tasks, such as switches, routers, and firewalls, across the environment, VMware's virtual networking and security software allows essential business and data processes to run as efficiently as possible without compromising security or dependability. The following three examples used in SDN are:

- Microsoft's Virtual Machine Manager

The infrastructure of SDN can be deployed and managed using Microsoft's virtual machine manager (VMM). It offers a uniform administration experience and is employed to set up traditional data centres. VMM offers virtualized hosts, network and library resources, and allocated storage, among its capabilities.



Users can carry out a wide range of tasks when SDN is integrated with the VMM. These include designing and administering virtual network policies, guiding traffic flows between virtual networks, and managing the infrastructure, which includes network controllers, software load balancers, and gateways. It also incorporates a wide range of technologies, including software load balancing, the RAS gateway, and the network controller.

- VMware NSX

With more than 140 apps moved over the course of five months to an SDN architecture, Tribune Media has likely the largest SDN deployment utilising VMware NSX. Tribune Media was split off from the rest of the Tribune Company in 2012. As a result, the organisation had to upgrade its IT systems and apps. As an outcome of this, Tribune Media decided to use VMware SDDC as the basis for its IT infrastructure. A virtual networking and security programme called VMware NSX is used in Software Defined Data Centres (SDDCs), which offer cloud computing based on VMware network technology. Switches, routers, and firewalls are just a few examples of the network functions that are distributed throughout the environment by NSX using a network hypervisor. Because of its agility, flexibility, and security, Tribune Media chose VMware's NSX.

- The Forefront of Tech Innovation

SDN can be implemented in many different ways. Even though the technology is still in development, significant service providers are using it. SDN, which will power future technological advancements in networks, promises to lower operational costs and offer more precise security.

# **Chapter 2**

**Architecture of Software  
Defined Network and  
OpenFlow Protocol**

## **2 CHAPTER**

# **Architecture of Software Defined Network and OpenFlow Protocol**

### **2.1 Architecture of Software Defined Network**

A flexible model for networking is the Software Defined Network. Compared to a conventional network, this one offers the following features: programmability, greater adaptability, increased flexibility, simple management, adjustability, and dynamic reconfiguration of network elements. By separating the control plane from the data plane, software defined networks improve network efficiency and offer two primary advantages: Network devices only function as a straightforward forwarding element in the data plane, and the controller has full control over the network's logic.

Software Defined Networking provides an innovative paradigm of networking that can help to fulfill the requirement of the user on demand. Moreover, SDN improves network control and permits the network provider to respond to changing business requirements [6-19]. The novel paradigm of Software Defined Network follows a layered structure given in Figure 2.1; SDN architecture consists of three layers that follow a bottom-up approach/strategy. Each layer has been designed for a definite purpose.

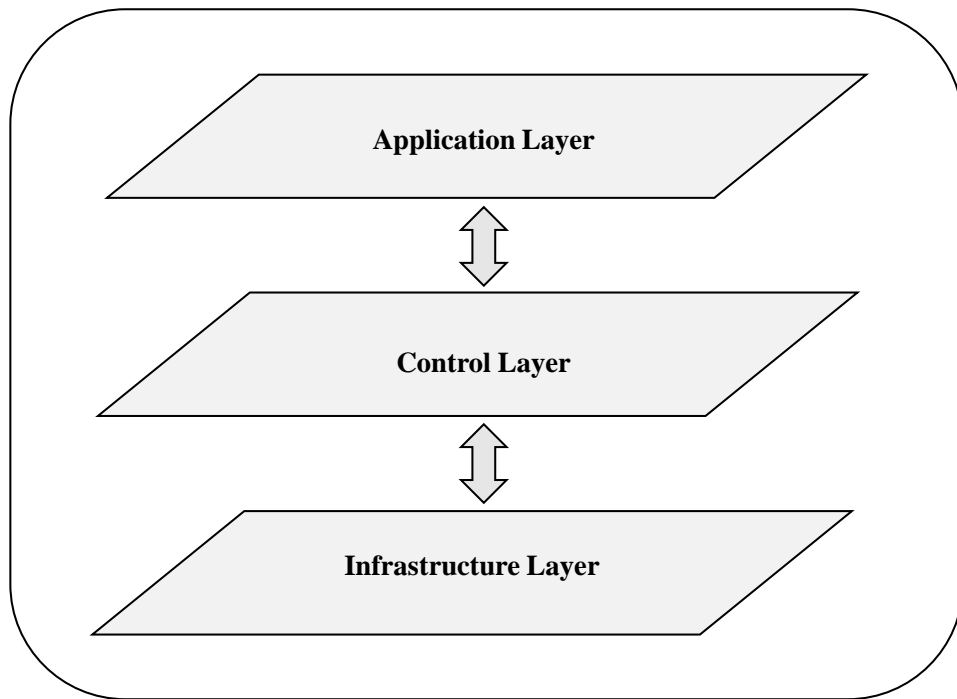


Figure 2.1: Layer Structure of Software Defined Network

The controller controls and directs the behaviour of all forwarding components, which are present in the bottom or infrastructure layer. They serve as a straightforward sending element as a result. Because it must forward data, gather statistical data, and monitor or watch, it is also referred to as the forwarding plane or data plane. The "Brain of the Network" [2,6-19]—also known as the control layer—follows, offering a logical overview of the complete network. It is commonly referred to as a controller. By dynamically modifying the policies of the network components, the controller can set or reconfigure them. Between the application plane and the data plane is the control plane. In the application layer, various network applications are implemented to control the logic of the network domain. These programmes are constantly running or operational on the controller's top. The controller manages three types of APIs such as Eastbound-Westbound, Northbound, and Southbound. The East-Westbound APIs are used when multiple controllers are communicating. The northbound APIs, like the REST APIs, allow for communication between the application and control plane; the southbound APIs,

like the OpenFlow protocol, allow for communication between the data plane and control plane.

The architectural design of Software Defined Network has mainly three planes: the data plane, the control plane, and the application plane, as shown in Figure 2.2. Each plane has its own specific functions [1-9,16,64,66].

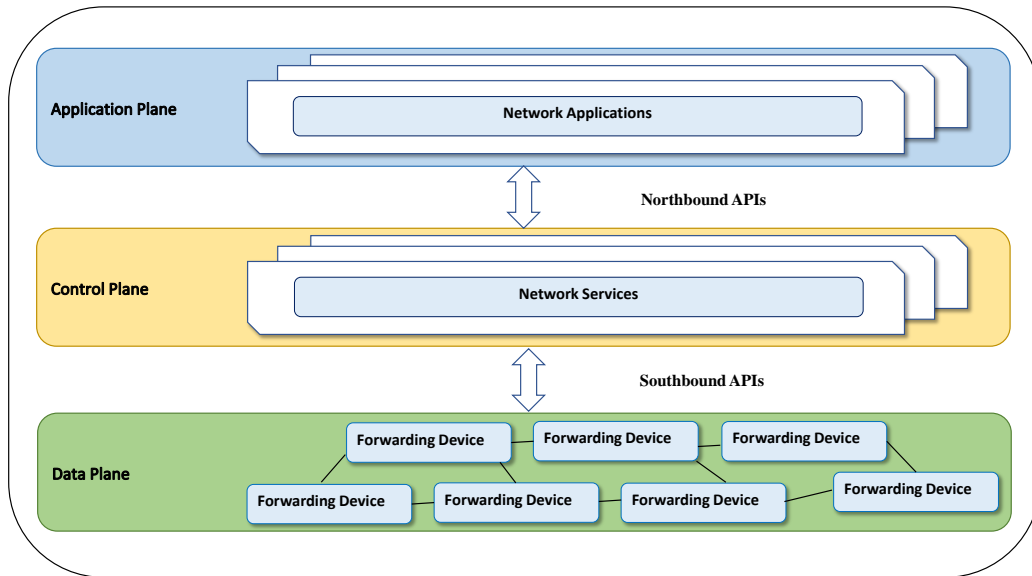


Figure 2.2: Architecture of Software Defined Network

## 2.2 Data Plane

The majority of the data layer is made up of forwarding devices like switches and routers. Devices in the data plane that carry out different network operations, like packet forwarding, can be either hardware or software-based. These forwarding elements have well-defined flow rules which aid to take action on the incoming packets in the network like forwarding to a specific port, dropping a packet, forward to a controller. These instructions are installed by the controller through southbound interfaces in the forwarding elements. These devices function as a straightforward forwarding element whose behaviour is dedicated by the controller as a result of the decoupling of the control plane from the data plane. A switch checks the header field of a newly received packet against the flow rules install in the flow table. If any appropriate flow rule matches, then the corresponding action takes place. Otherwise,

send a PACKET-IN message to a controller then the controller installs new flow rules in the table. The data plane is also known as the infrastructure layer. It is a bottom layer of the Software Defined Network paradigm; A major duty of the data plane is data forwarding, as well as monitoring information and gathering statistics.

## **2.3 Control Plane**

The control plane, also known as the controller, which offers a centralized, global perspective of the network. It serves as a "Brain of Network" because of this. The controller can modify the forwarding components' configuration by dynamically modifying their policies. Between the application plane and the data plane, the control plane acts as a mediator. It can interact with the application plane using northbound interfaces (northbound APIs), and it can communicate with the data plane using southbound interfaces (southbound APIs). Most controllers used OpenFlow as a southbound API. A controller has network management ability and resolves network-related issues by providing a logically centralised view of control given by the Network Operating System (NOS). So, the developer does not need to maintain information about how data is distributed at a low level among these forwarding elements while defining network policies. In order to implement network intelligence, an SDN controller manages network applications and business logic through northbound interfaces; to control the flows of the switches/routers in the data plane through the southbound interfaces. The flow of southbound API can be going down and the northbound interfaces can be going upward in the SDN architecture. The controller has the responsibility how maintaining and upgrading the topology information of a network. Through programmability, the controller can control the network traffic and decreases the complexity network. It is done by the controller due to a centralized logical view of the network.

## **2.4 Application Plane**

Different network applications and services are implemented in the application plane to manage the logic of a complete network domain. These programmes are always active on the controller's topmost layer. Northbound APIs, which seek network state and offer a facility to manipulate the services, make it feasible for the application

and control plane to communicate. Every instruction of applications will be translated into forwarding elements of the data plane through southbound APIs (mainly used the OpenFlow). Most northbound APIs are REST (REpresentational State Transfer) APIs. Through this plane, network users can control and manage the network. Because these applications define policies that are implemented by a controller via southbound interfaces in forwarding elements.

So, to define SDN as an architecture of networking that works on these four pillars are (as shown in Figure 2.3):

1. To remove or extract the control capability from network devices by decoupling the control plane from the data plane. As a result, these devices serve as a simple forwarding element.
2. Instead of destination-based, the forwarding decisions are flow-based.
3. The control logic is moved or transferred to an external entity known as a controller.
4. The network is programmable via applications run on top of the controller. The controller interacts with the underlying network elements of the data plane via APIs. It is an essential characteristic of SDN.

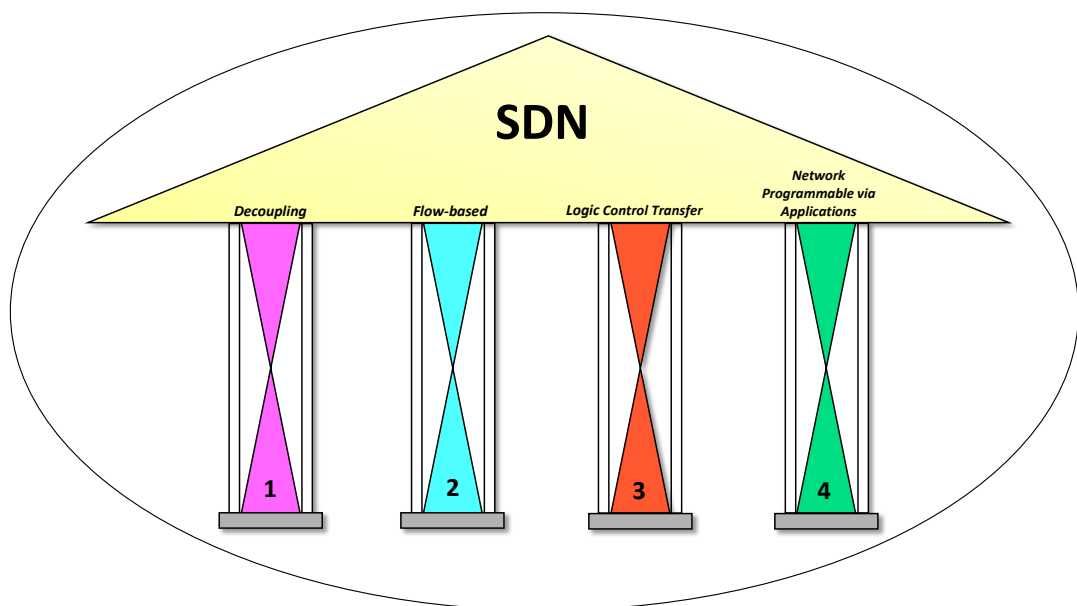


Figure 2.3: SDN Paradigm work on these Four Pillars

## 2.5 OpenFlow Protocol in SDN

One of the most well-liked and well-known southbound SDN APIs is the OpenFlow Protocol. It is an intermediary between the control plane and the data plane; it is used to make communicate among them. Various OpenFlow versions are available, but OpenFlow 1.3 is a widely supported version. The channel of communication between the controller and switch is established by using the OpenFlow protocol. they must have the same version of the OpenFlow protocol. For packet processing, OpenFlow 1.1 made two significant changes: a group table and a pipeline of multiple flow tables. Because of this, OpenFlow V1.0 and V1.1 are incompatible with one another. The architecture of the OpenFlow Protocol shows in Figure 2.4.

The OpenFlow switch has mainly two constituents [6,8,15-17,27]: -

1. **Secure Channel:** Establish an encrypted link between the controller and switch using SSL and TLS.
2. **Flow Table:** For the purpose of handling the incoming packets, all forwarding rules are implemented in the flow table.

## 2.6 Flow Table

A flow table with a number of flow entries is present in every network node or element. The controller provides these rules, which the OpenFlow switch uses. To each OpenFlow switch has at least one flow table. These flow entries are organized by their priority in a flow table. Each flow entry has primarily three fields as shown below in Figure 2.5.

- **Matching/Header Field:** To check or match the source address, destination address, port, IP address/MAC address, and VLAN ID against the header information of the incoming packet.
- **Actions/Instructions:** To specify the course of action for matching packets, such as packet forwarding, dropping, modification, etc.
- **Stats:** To show statistical information about the flow like the number of received packets, bytes, and duration of the flow.



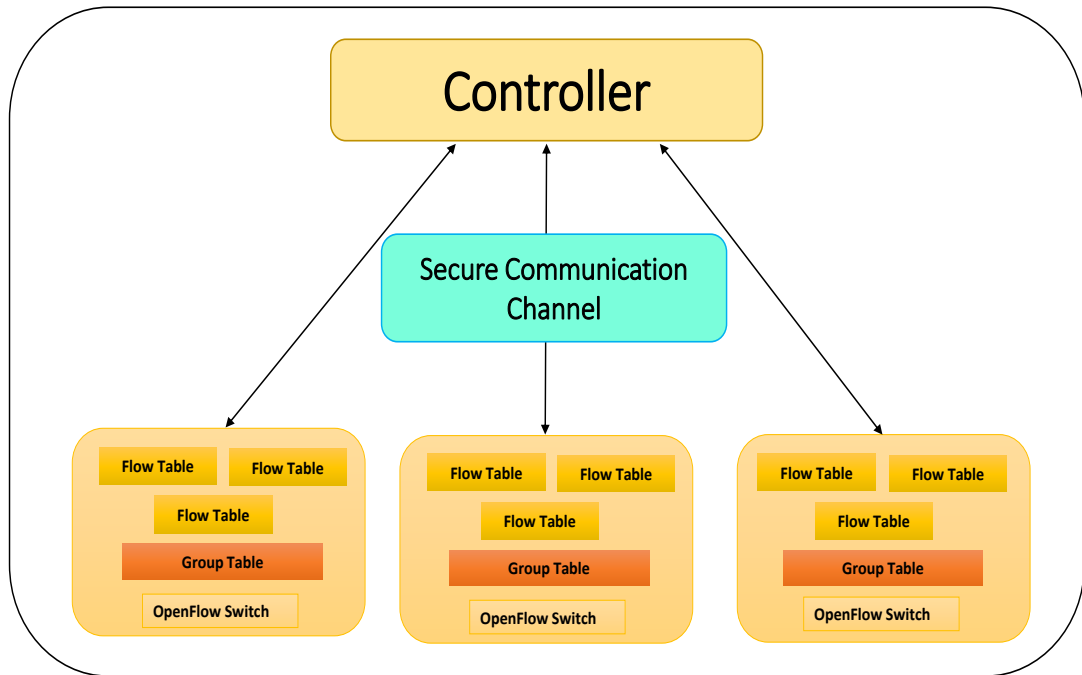


Figure 2.4: OpenFlow Protocol Architecture

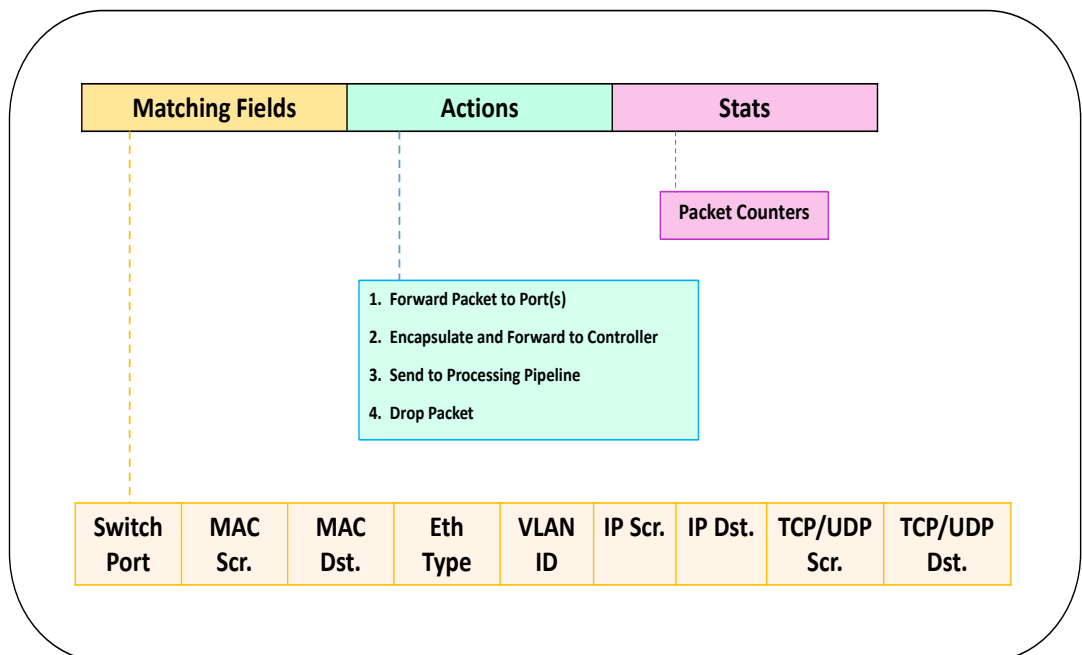


Figure 2.5: Format of the Flow Entries

The controller has the authority to situate and operate the flow entries in network devices. A controller has privileges to insert, delete, and modify the flow entries in a table and instruct the switches on how the packet should be transmitted in the data plane. Therefore, a controller can easily manipulate the forwarding behavior of these devices.

### **2.6.1 Type of Message**

For this purpose, the OpenFlow protocol supports numerous kinds of a message which define how to coordinate these forwarding devices in a network are listed below [6,8,16]:

- **HELLO:** - To set up a connection between controller and switch by using the HELLO message.
- **ECHO:** - To check the liveness of connection and its operational status between the controller and forwarding devices. An ECHO-REQUEST message is sent by a controller to a switch, and a switch responds with an ECHO-RESPONSE message to the controller. When a switch fails to respond to an ECHO-REQUEST message then identify that connection is lost.
- **PACKET-IN:** - When a switch can't find a suitable flow rule in a table, it sends a PACKET-IN message to a controller.
- **PACKET-OUT:** - The PACKET-OUT message is sent by a controller to a switch in reaction or response to the PACKET-IN message. The PACKET-OUT message specifies how to deal with these packets.
- **FLOW-MOD:** - A controller can update the flow entries of switches by using the FLOW-MOD message.
- **PORT-STATUS:** - A switch sends a notification to a controller by using the PORT-STATUS message. There is a change in the status of ports such as port-up or port-down. This information helps to manage failure in the network by a controller.

All messages are grouped into three types of messages controller-to-switch message, asynchronous message, and symmetric message in Figures 2.6 (a to c):

- A. **Controller-to-Switch:** These messages, such as PACKET-OUT, FLOW-MOD, ROLE-REQUEST, etc., have been initiated by the controller to directly examine the status of the switch.

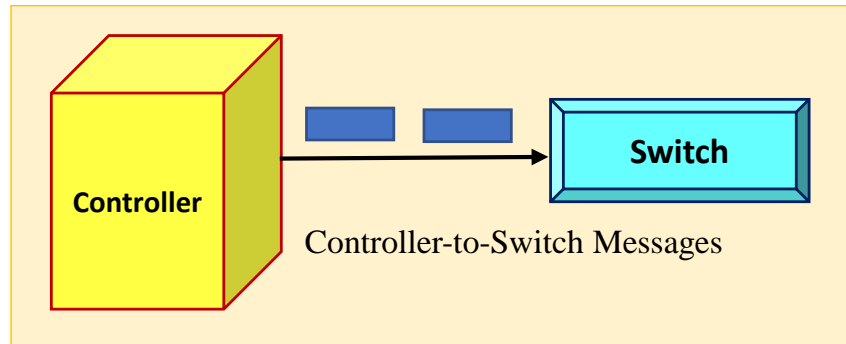


Figure 2.6 (a): Controller-to-Switch Messages

- B. **Asynchronous:** These messages are originated by the switch to inform the controller about the network change that an event occurs by a switch like PACKET-IN, PORT-STATUS, etc.

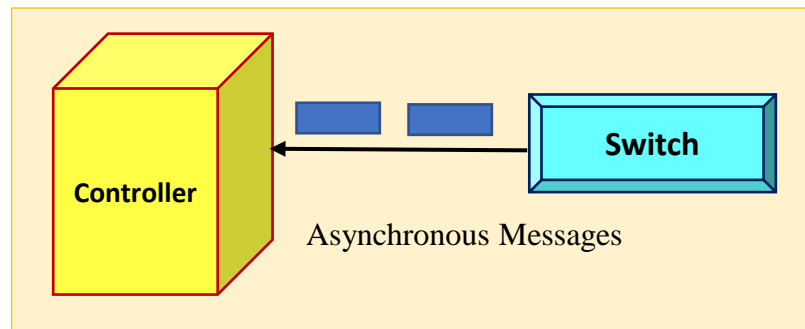


Figure 2.6 (b): Asynchronous Messages

- C. **Symmetric:** Symmetric messages are beginning either by the controller or switch to send each other without any solicitation such as HELLO or ECHO messages.

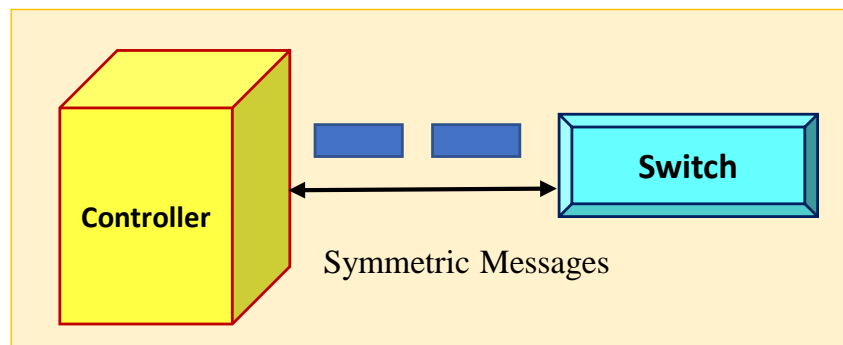


Figure 2.6 (c): Synchronous Messages

## 2.7 Implementation of Flow Rules

The two different techniques are used for implementing flow rules in a table in either a reactive manner or a proactive manner by an SDN controller. The working of these methods is as follows and the comparison between both methods is highlighted in Table 2.

### 2.7.1 Reactive Method:

When a host sends a new packet in the network then the switch matches its header field against the flow entries of the table. If the header field of the packet matches then the corresponding action is executed otherwise, the switch sends the PACKET-IN message to the controller. The controller inserts or alters flow rules by using PACKET-OUT, and FLOW-MOD messages. After that switch can easily forward packets to the desired destination [6-27, 64,66] shown in Figure 2.7.

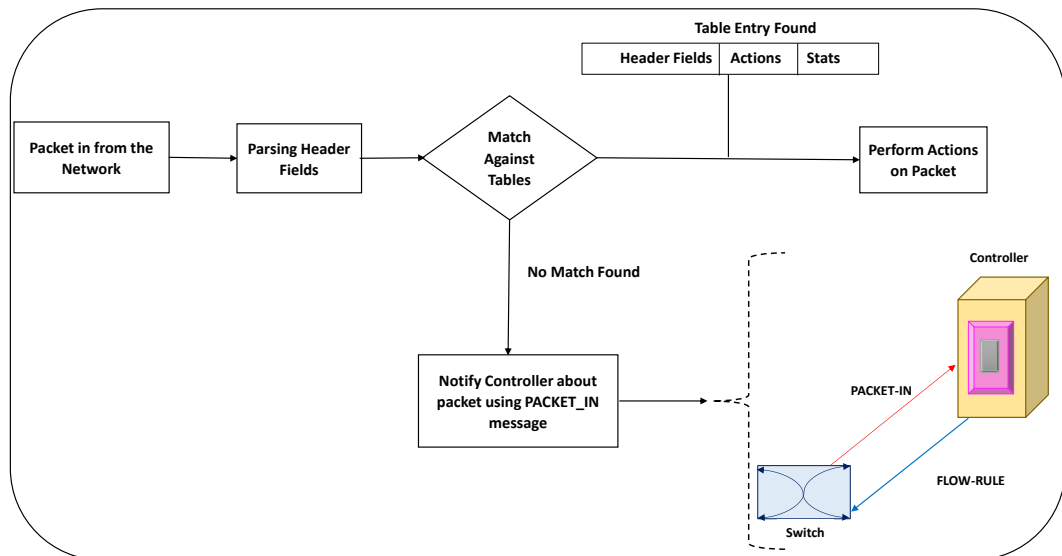


Figure 2.7: Packet Forwarding Process in Reactive Method

When a host sends a new packet in the network then the switch matches the header field of the incoming packet to the corresponding flow entries in a flow table. If the header of the packet is matched then the corresponding action is executed; otherwise, the switch sends a PACKET-IN message to the controller when the header of a packet is not matched. The controller installs/inserts new or modified flow rules in a flow table by using FLOW-MOD or PACKET-OUT messages in switches of

networks. After those switches can easily forward the packets to the desired destination. These steps are highlighted in Figure 2.8 by using numbers 1, 2, 3, and 4.

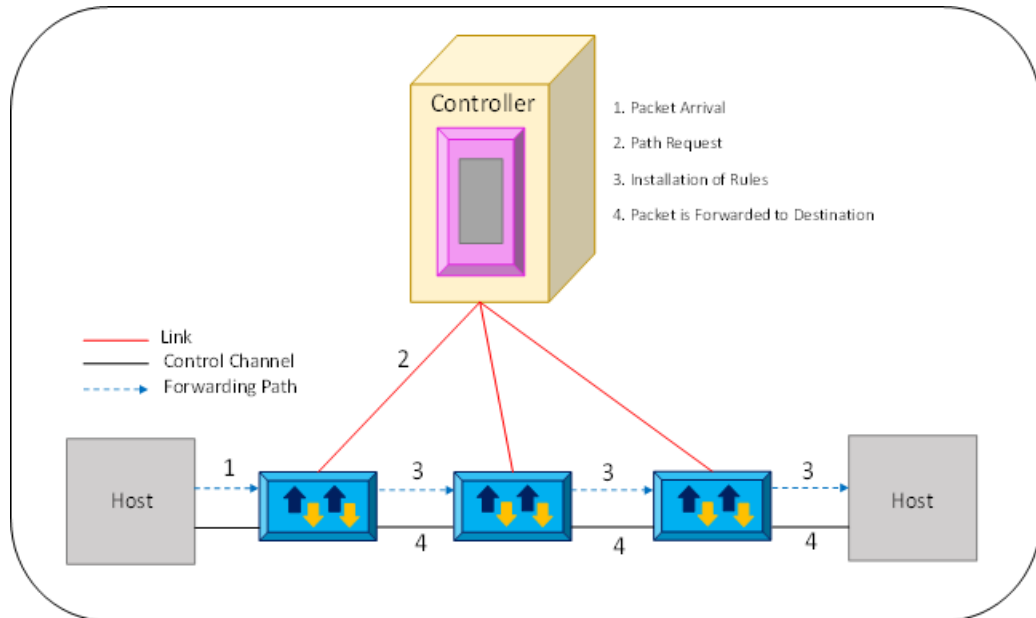


Figure 2.8: Notify to the Controller by using PACKET-IN Message

### 2.7.2 Proactive Method:

A proactive method controller previously installed flow rules in a table for managing network traffic. The PACKET-IN event never happens in this method because the controller can populate the flow rules before the packets arrive. So, the latency of a network is eliminated that brings due to the involvement of a controller in the PACKET-IN message. The value of the action field is set to FLOOD always in the proactive method.

Table 2: Difference between Reactive and Proactive Method

Reactive Method	Proactive Method
The reactive method installs the flow rule after the fault occurs or sends a new packet by the host.	The proactive method installs the flow rule before the fault occurs or a new packet sends by the host.

The switch sends the PACKET-IN notification to the Controller if the packet's header cannot be matched.	No PACKET-IN event occurs in the proactive method.
Reactive Flow introduces latency due to the involvement of the controller.	Proactive Flow eliminates latency which involves due to the controller.
Recovery time is more than the proactive method.	Recovery time is less as a compared reactive method.

## 2.8 OpenFlow Pipeline and Group Table

OpenFlow 1.1 [6-16], which is used for packet processing, introduces two major changes: a group table and a pipeline of numerous flow tables. For this reason, the OpenFlow V 1.0 and OpenFlow V 1.1 versions are not compatible. In OpenFlow 1.1 version offers a concept of the pipeline for multiple flow tables. When a packet has arrived in a pipeline then the metadata field of a packet is used for the matching process. Metadata of the packet is used from one step of the pipeline to the next step of the pipeline till the execution of the action set is occurring. This action set is also known as the instruction set of a flow table. The list of action sets is given below in Table 3. And the working/processing of the pipeline in OpenFlow is shown in Figure 2.9.

Table 3: List of Action Set in OpenFlow

Action Set/ Instruction	Description
Apply-Actions	Apply action immediately without any modification to the action set.
Clear-Actions	Clear the entire action list.

Write-Actions	Merge the given action(s) into the current action set.
Write-Metadata	Update the Metadata field.
Goto-Table <next-table-id>	Goto next table in the processing pipeline. A table-id of the next table is greater than the current table-id.

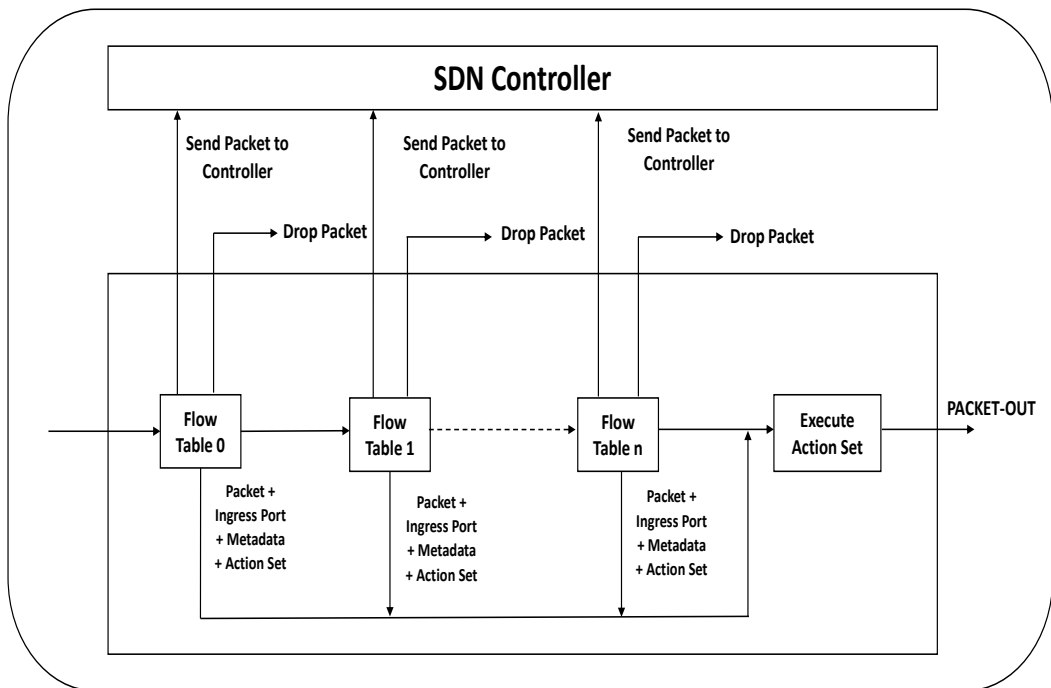


Figure 2.9: Packet Flow Process in the Pipeline of OpenFlow

OpenFlow V 1.1 and its later versions also support the group table feature. Each group table has included these fields such as group-identifier, group-type, counters, and action-buckets in the group table as shown in Figure 2.10.

Group Identifier	Group Type	Counters	Action Buckets
------------------	------------	----------	----------------

Figure 2.10: Format of Group Table Entries

Every group table entry has a unique group identifier in the table. The same group identifier is used by several flow entries in a table; A group type specifies which type of a group is applied. The purpose of these group types is a list in Table 4.

Table 4: Various value of Group Type in OpenFlow

Sr. No.	Group Type	Meaning
1.	All	This type of group is used for “broadcast” or “multicasting.” In this group, packets are processed in all action buckets.
2.	Select	It uses a selection algorithm to run selected action buckets.
3.	Indirect	This type of group allows different flows and groups which is related to a common group.
4.	Fast Failover	It is used to implement a backup path in a switch to measure the liveness of the port.

The counter field is used to collect statistical information on packets processed by a group. The action bucket field of the group table contains a set of actions whose execution depends on the type of group. In OpenFlow 1.3 version introduces a meter table; that is directly connected to the flow table through a meter identifier. The second field is the meter band that specifies the rate limit (data rate) of a packet. If the rate limit of packets is exceeded then drop the packets. The counter field provides statistical information on packets. The format of the meter table entry is shown below in Figure 2.11.

Meter Identifier	Meter Bands	Counters
------------------	-------------	----------

Figure 2.11: Format of Meter Table in OpenFlow 1.3

The OpenFlow protocol prefers to use TCAM memory for the purpose of storing information about flows in a table because it provides flexibility and efficiency in terms of matching capabilities. Its size and price are both astronomically expensive, though. TCAM memory is insufficient to keep a large number of flow entries in the flow table concurrently [6,816]. In order to enable the flows to expire or be



withdrawn after a set amount of time, OpenFlow offers a flow timeout function. For this purpose, two fields available are:

- **Idle Timeout:** The flow will expire if it is inactive for the allotted amount of time.
- **Hard Timeout:** A flow entry must end after a predetermined period of time (number of seconds). It doesn't matter whether packets are striking or matching an entry.

In both fields, timeout's default value is 0. These flows are ongoing and cannot be stopped. That is why the OpenFlow protocol is regarded as the father of Software Defined Networks due to its broad acceptance and smooth deployment of SDN.

## **2.9 Summary**

Software Defined Networking offers a cutting-edge networking architecture that can assist in meeting the needs of the user as needed. SDN also enhances network control and gives the network provider the ability to adapt to shifting business needs. The three layers that make up the unique SDN paradigm are built from the bottom up. Using the four pillars as the foundation of its architecture. The OpenFlow Protocol is among the preferred and well-known southbound SDN APIs. Using the OpenFlow protocol, a communication path is built between the controller and switch.

# **Chapter 3**

## **Literature Review in Software Defined Network**

# **3 CHAPTER**

## **Literature Review in Software Defined Network**

### **3.1 Introduction**

Today computer networking is the main source of communication in which different persons exchange and share their ideas, develop and improve their skills, to get a new insight of knowledge and experience on challenges that occur in their work. In the last few years, computer networking has rapidly increased the demand for services; but it is very difficult to cope with the demand of requirements of the user in the traditional networks. To cover this situation, it is required to restructure the traditional network to increase the efficiency of a network that can fulfill the requirements of a user as per their demand. It is possible through a novel and innovative paradigm of networking which is known as Software Defined Network. This chapter provides an exhaustive review of the literature on the framework for Software Defined Network and its functionalities. Finally, it summarises a review of the literature on SDN controllers and their features before going into more detail about how the control plane and data plane interact via the southbound application programming interface, like the OpenFlow protocol. Discuss the related study on fault tolerance and load balancing in SDN networks as well.

## 3.2 Traditional Networks

Due to the ongoing development of technologies, computer networks have a significant impact on how humans' lifestyles change through time. However, adding new functionality to traditional network devices is a challenging and complex operation in traditional networks. Due to the traditional network's lack of programmability, it is an uneconomical duty or task since it raises the network's operational and capital costs. Additionally, all network devices must upgrade their hardware if any changes are attempted to the control plane of the conventional network. Vertical integration between the two functional components is the cause of the rigidity of traditional networks.

**S. Haji** *et al.* [1] in traditional networks both functional components are vertically integrated. This vertical integration, it makes more complex, complicated, and difficult for the network operator to configure or reconfigured the network devices; this is happened due to the predefined procedures that responded to a fault and load modifications in the network.

**D. Gopi** *et al.* [2] in conventional routing protocols have a very intricate and rigidity of adaptability in the network change. This happened due to a tightly coupled bond existing between both functional components. In a traditional network, every router can recompute the alternate paths independently in the network. After acquiring the updated knowledge and building a new routing table based on it. Once the convergence procedure is finished and the tables have been updated with the modifications. Then, the network's packet transport is restarted from the source to the destination. For this reason, convergence time has a great influence on the change of topological size in the conventional or traditional networks; because all the routers send updates through the BGP protocol to up-to-date their routing (RIB and FIB) tables. Consequently, in order to expand the topology size in conventional networks, the routing time required for convergence is continuously increasing; because the information of a failure of a link or node is propagated or broadcasted via flooding to update all the routing tables of routers in network. But, in case of SDN networks, there is no need to propagate the updated information to all the forwarding elements of the networks through flooding. The controller recomputed the alternate path and

updated this information only on the affected switches rather than all devices of the networks. It is the responsibility of the controller to perform the routing convergence and maintain topology knowledge of the network; it provides a centralized logical view of the entire network. Thus, the routing convergence time in Software Defined Networks is better than the conventional networks.

**M. J. Anjum et al.** [3] the traditional Data Center Networks (DCN) approach is insufficient for the amount of data transfer that is presently occurring in a data centre network. It is unable to utilize all the resources. The novel paradigm of Software Defined Network (SDN) can be implemented in DCN networks to make better network management compared to the traditional network because the control plane and data plane are separate. The SDN uses multiple paths in a data center network for transmitting the data; the other remaining links are used to regulate the data in the network in a parallel manner. It enhanced data transmission in terms of throughput and packet loss during link breakdown.

**G. Khetrupal et al.** [5] In traditional routing protocols, select a single best route for a specified set of destinations by the decision selection process of the route; whereas in SDN OpenFlow based routing select the multiple routes to the same destination. Moreover, in SDN reconvergence time is better than in traditional routing protocols.

### 3.3 Software Defined Networks

This fact is well known that every day there are a lot of innovations and enhancements of the existing technologies. Software Defined Networks (SDN) responds as soon as possible to the network users' changing demands for the network's resources. As a result, it is possible to assert that network advancements position the SDN as networking's future. Software Defined Networks offer a new paradigm for networking innovation by dividing the control plane from the data plane. The goal of a Software Defined Network is to manage the network in a way that allows service providers and commercial entities to rapidly adapt to changing business circumstances. The capability to programme, alter, and dynamically reconfigure networking devices is provided by software defined networks. As a

result, SDN provides flexibility, and centralised view control reduces both the complexity and the expense of the network systems.

**D. Gopi et al.** [2] conventional routing protocols have narrow adaptability to network changes. But, SDN provides an effective method for this problem due to a centralized controller. In a conventional network, the convergence time is continually increased as increase the topology increase. In SDN, when a topology is scaled then the convergence time of routing has no significant change because all work performs by an SDN controller. It can send updated information only to affected switches rather than all devices of a network. Thus, SDN networks have less convergence time as compared to conventional networks.

**D. Kreutz et al.** [6] provides an in-depth examination of Software Defined Networks. A digital society is created via the internet when everything is connected and accessible from anywhere. The traditional network is difficult to maintain and would require wider adoption. Try reconfiguring the forwarding devices to respond to a fault, imbalance, or change because it is difficult to set them in accordance with predefined policies. It happened due to a strong bond existing in both functional components. Even this vertical integration makes it more complicated to reduce the flexibility and evolution of the network. Software Defined Network (SDN) provides an emerging paradigm of networking that gives hope to changes in a traditional network. First, break the vertical integration between both components. Second, this separation makes all forwarding devices a simple forwarding element in the data plane, and control logic is implemented in a logically centralized controller. Thus, a controller has direct control over these data elements through a well-defined application programming interface (such as OpenFlow). SDN architecture follows a bottom-up approach. It offers an opportunity to solve long-standing problems in a traditional network.

A global perspective of the network, dynamic programmability in forwarding elements through APIs, and different applications running network logic on top of a controller are some of the key concepts offered by SDN. Compared to traditional networks, these are much easier to build and deploy. The ongoing research and

challenges in SDN are fault tolerance, load balancing of multiple controllers, scalability, synchronous, and so on.

**H. Zhang** *et al.* [7] When comparing the performance of SDN routing to legacy routing protocols, the legacy network has faster packet forwarding than SDN when the network scale is lower. SDN speeds are faster than traditional networks when the network scale is bigger because switches must not maintain all network information in SDN.

**S. Khan** *et al.* [9] generally in SDN routing is implemented in the control plane; because all information is maintained on the control plane like network state and topology. The most realized issues in SDN are the single point of failure and scalability. To resolve these issues, then there is a vast space for development that can fulfill the requirements of SDN.

**I. Radu** *et al.* [10] describe the behavior and compatibility of SDN components (software elements) connected to traditional hardware equipment (hardware elements); because an SDN controller has the privilege to monitor all nodes of a network and manage network traffic according to the network load. There are no special configurations required in hardware networks or software networks. But compared to the hardware switch, the virtual switch introduces a three times smaller latency. Additionally, SDN components are not just pieces of hardware that can be used virtually. Many companies offer 100% virtual solutions for various communication service kinds. The architecture of these solutions is the same as that of hardware devices.

**Z. Hu** *et al.* [11] enables the administrator to configure the resources of the network in SDN very quickly. It is also a facility to adjust the traffic flow dynamically in the network due to a controller. It can help the developer to implement security functions that offer security services in the network when developing an SDN controller.

**I. Bedhief** *et al.* [12] about billions of smart heterogeneous devices are connected via communications technologies. The SDN is an emerging network architecture whose aim is to improve network performance through the global vision of a network that is offered by the controller.

**C. Tselios et al.** [13] Software Defined Network simplifies the policy enforcement or implementation in the network elements. As a consequence, it becomes easy to reconfigure the network elements dynamically rather than an individual device. The configuration of the network element can be changed by a controller. A solitary SDN controller presents a failure risk for this reason.

**Y. Yu et al.** [15] Software Defined Networking provides a paradigm of a network that offers simplified network management and innovation in networking. But still, give more attention to the reliability of the network. The reliability of SDN is ensured via fault tolerance. System monitoring, fault diagnosis, fault recovery, and fault repair are the four main tasks that comprise the fault tolerance procedure. SDN controller are still in the early stages of developing fault tolerance. So, it is considered a future work for research in SDN.

**A. Malik et al.** [16] Software Defined Networks is a hot and burgeoning topic that has attracted more attention in both the commercial and academic sectors; because it decouples the control plane from the data plane. Moreover, the SDN paradigm provides these features like programmability, adjustability, and dynamically reconfiguration of forwarding devices in the network. But fault tolerance and recovery are some of the key issues that SDN faces. Moreover, only 4% of the research effort contributes to fault tolerance in SDN and the rest in other fields. To better SDN traffic engineering and increase network efficiency, a controller offers a global view of the network. To address these issues with network scalability and resilience, fault tolerance requires more attention. If something goes wrong in one layer, it may or may not affect the other layers. Because of this, each layer of SDNs must be built separately to deal with the problems of fault tolerance on that layer. A connection failure, for instance, might not have an impact on the control layer, whereas a controller failure might have an impact on the entire infrastructure layer.

**M. R. Parsaei et al.** [17] compare current IP networks with SDN networks. In the current network combining both components; Consequently, due to predetermined rules, network configuration is challenging. Try to adjust them so they can react to flaws, loads, and modifications. However, a new style of networking separated both functional components. As a result, it provides smarter, flexible, and controllable



management facilities in the network. After a study on fault tolerance in both networks to analyze the current network most challenge is the time to repair and update the network whereas the SDN network increase computational overheads in the controller. So, try to develop an algorithm for fault tolerance on the criteria of the lowest packet loss in the network.

**J. Chen** *et al.* [18] proposed a significant component that is OpenFlow which is useful for failure detection and recovery because SDN is unable to survive in a large-scale network when facing a failure. So, to design a mechanism for achieving fault tolerance. Then compare both reactive and proactive methods respectively; Every time a controller fails, the OpenFlow channel and the supporting switches go out of control. If using more than one controller in the control layer results in a more stable system. For this reason, the author lights this issue (fault tolerance) in SDN as a further research direction.

**B. Isong** *et al.* [21] a single controller controls all resources in a network and then poses a single point of failure. It has a direct impact on the scalability, reliability, availability, and performance of the network. The likelihood of a network breakdown brought on by a single controller rises as network complexity rises. A system's fault tolerance is its capacity to keep working in the face of hardware or software breakdowns. Its nature is unreliable. The three steps of fault tolerance are fault detection, recovery planning, and recovery execution. To ensure that the single point of failure is eliminated, the fault should be identified and recovered as early as feasible. This is due to the fact that the controller is a crucial component of the network, and having only one faulty controller will have a negative influence on the network's availability, scalability, and dependability/reliability.

**S. Vissicchio** *et al.* [22] SDN controller exerts a centralized logical control of a whole network that make simplify network management and increases the efficiency of a network. But, SDN comes with its own set of issues in terms of resilience, robustness, and scalability.

**I. F. Akyildiz** *et al.* [24] SDN paradigm promotes innovation and evolution in networking which improves resource utilization, simplifies network management, and decreases the operating cost of a network. Traffic engineering receives a fairly

little amount of research attention compared to the development of SDN architecture. For SDN to improve traffic engineering, this viewpoint, including flow management, fault tolerance, load balancing, topology updating, and traffic analysis in terms of scalability, availability, dependability, consistency, and accuracy, must be accounted for.

**M. Casado** *et al.* [25] and **O. Akonjang** *et al.* [26] in early 2000, the demand for the Internet is grown rapidly as well as; they needed more flexibility in the networking services such as reliability. By dividing the data plane from the control plane, several attempts have been made to attain originality in the networking industry throughout this time. The 4D project was used to reorganize the network design into four planes, including the Decision, Dissemination, Discovery, and Data planes. Two components that Ethan project (a SANE extension) uses 4D features are a centralised controller that oversees the network's overall security procedures and Ethan switches that have gotten forwarding instructions from a controller. The first of Ethan's two key drawbacks is its inability to comprehend the absence of network nodes and users, while the second necessitates manual flow level routing management. Ethan was not adopted as a result. In 2006, proposed a clean-slate security architecture (SANE) to control security policies in a centralized way instead of accomplishing it at the edge. Designing a new architecture with security is a fundamental goal. Its main goal is to establish an architecture that supports simplicity, implements security at the link layer, and hides all topology and service information from unauthorized parties. The SANE and its replacement Ethan provide a base for the beginning of OpenFlow. At Stanford University, a research group has developed the OpenFlow protocol as a clean slate technology. Communication between the both plane is made feasible by one of the most well-liked southbound interfaces in SDN. The concept of the Software Defined Network was created by the OpenFlow initiative.

**M. Paliwal** *et al.* [28] Because it handles all control decision responsibilities while routing packets in a network, a controller in SDN architecture serves as the network's brain. As a result, centralised decision-making capacity improves network

performance. A centralised controller, however, is unable to solve the scalability problem.

**A. Nantoume** *et al.* [29] the fact, that operators are refusing to deploy their equipment in the unprofitable region because the CAPEX and OPEX of equipment are relatively high. As a result, Software Defined Networking has decreased a network's CAPEX and OPEX while also enhancing the quality of service (QoS).

**S. Scott-Hayward** *et al.* [33] increase innovation in networking applications through Software Defined Networking which aids to reduces the cost of a network. This paper highlights the characteristics of SDN. Further, it exploits characteristics of SDN that may be more secure than traditional networks.

**S. Narayana** *et al.* [35] SDN programming monitors the flow of traffic in network paths by using queries through regular expression; it can be easily converted into DFA to a representation of the path of queries in SDN.

**R. Wang** *et al.* [36] dedicated load balancer becomes expensive due to congestion and quickly arises a single point of failure. Network switches can use either microflow or wildcard rules to split traffic among server replicas according to packet handling rules set up by controllers. because an immense amount of flow entries in the flow table cannot be stored in TCAM memory. The OpenFlow install flow timeout feature allows the flows to expire or be deleted after a predetermined period of time. So, without interrupting current connections, immediately adjust the changes in load balancing rules and regulations.

**X. Xu** *et al.* [37] traditional network architecture has become complicated to manage due to a lack of programming capabilities; it also increases the deployment cost of the network as compared to SDN.

**W. Liao** *et al.* [38] by separating both layer in SDN changes the limitations of the existing network. This study suggested a load balancing mechanism that decreases response time by dynamically adjusting the flow rules in a table; it can dynamically transfer the extra load to a server that is under load.

**S. Bera et al.** [41] Path estimator and flow manager are two components that are suggested for use in the Software Defined Networks implementation of flow tables. This scheme improves network performance by nearly 50% approximately.

**T. Luo et al.** [44] identify the main two problems in the network that are rigid to policy changing and difficult to manage which rise technical challenges in both planes such as creating flow, and overhead introduce when a load of network traffic is high.

**K. Wang et al.** [46] when network coverage has increased then a single controller faces a bottleneck problem in the network; due to a scalability issue in the SDN. As a consequence, network performance degrades.

**M. C. Nkosi et al.** [48] the use of load balancing helps networks run more efficiently, are more available, and experience less gridlock. In legacy networks, a particular hardware device is required for costly load balancing. But, in SDN there is no need for a device for load balancing because the controller manages all calculations of an optimal path; and then configures the flow table in switches by using a group table.

**H. Kim et al.** [50] CORONET used the restoration technique for the failover mechanism to re-plan or re-compute the link after the failure. Each of CORONET's four primary components is in charge of a distinct job. Since the topology finding module is used to collect periodically information about a network which helps to construct a global view of the network. The route plan module is used to calculate the route path as well as the backup route in case of link failure by utilizing Dijkstra's algorithm. To enforce this routing route in the OpenFlow API, the VLAN switch configuration module is used to configure the switch port with the VLAN ID. The traffic assignment module is used to allocate traffic in a network. But CORONET suffers a high latency problem during installing the new flow rules because it must control a broad perspective of the network.

**P. C. Fonseca et al.** [51] issues with SDN fault management are presented in a thorough perspective. The placement of a controller, reliability, scalability, and failure of a controller are just a few of the problems that the conventional network

presents in comparison to a logically centralised control plane. These are ongoing research issues in fault management in SDN.

**F. Bannour et al.** [52] Software Defined Networking continuously gained popularity in recent years, in both study and academia. The centralization of an SDN controller offers new opportunities to make network management easy. But arises several issues which are related to the SDN control plane such as scalability, reliability, consistency, and interoperability.

**S. Bera et al.** [53] SDN controller includes two different management policies device management and network management which improve network performance. Therefore, the packet delivery ratio is higher than in traditional networks due to a global view of the SDN controller. To address an open issue in SDN is an optimal placement of the controller problem.

**G. Wang et al.** [55] SDN controller interact with the application plane via the northbound API, which facilitates network control and its services. One significant problem that occurs when using numerous controllers is the placement of the controllers. To resolve this issue must be a focus on minimum network latency, maximum reliability, and minimum deployment cost of a network.

**A. Gonzalez et al.** [58] SDN paradigm enhances innovation and flexibility in the network. So, robustness and fault tolerance must consider as the main criteria for networking. But there are still many open issues like consistency, durability, and scalability in SDN. The performance of the controller is classified into two concepts such as controller latency and controller throughput. Moreover, a single controller always has a threat of failure.

**L. Sidki et al.** [59] The network traffic is decided by the SDN controller, who has a wide overview of the network's capabilities. Support on a single controller is unsuitable due to two factors. Initially, the network encounters a solitary point of failure, or SPOF. Secondly, it increases the demand for network traffic to help deal with speed bottlenecks. Network throughput is reduced and network latency is increased as a consequence.

**N. Medhi** *et al.* [60] the perception of a Software Defined Network relies on a centralized controller. The various challenges addressed a centralized control architecture faces as scalability, availability, and fault tolerance in SDN.

**Y. E. Oktian** *et al.* [61] a survey on the design choice of SDN that may be influenced by these issues scalability, robustness, failure, privacy, and consistency.

**N. Katta** *et al.* [62] and **A. Mantas** *et al.* [63] Ravana required to be modified the switch and OpenFlow to be extended with hitherto unforeseen addition to the protocol. Rama does not require modification compared to Ravana; because Rama handles consistency by using the event processing cycle as a transaction. But the cost of this technique is increased and also incurs higher overhead compared to Ravana. These overheads also decrease the performance of the network.

**M. Karakus** *et al.* [64] traditional architecture network applications and devices are complicated to configure (or reconfigure) because they required highly skilled personnel. It is a costly and time-consuming job; It faces an obstacle of vendor dependency. SDN architecture dissociates the control plane from the data plane; It allows a network administrator to supervise network services via the abstraction of lower-level functionality. It is possible in SDN through a logically centralized technology. For this reason, especially a controller (control plane) suffers scalability issues in SDN which also affects the controller performance.

**O. Blial** *et al.* [65] by separating the control plane from the data plane, software defined networking offers a number of advantages. The biggest problem in SDN, however, continues to be SDN networks' scalability, dependability, and availability. Therefore, the centralised SDN design cannot fulfil the requirements for a network's efficiency, availability, and scalability.

**Y. Zhang** *et al.* [66] Software Defined Networking decouples the control plane from the data plane as a compared traditional network. SDN provides a programmability facility to configure the network elements. One of the major complaints against SDN is that it has a single controller failure point, which lowers availability and performance across the board. The performance and scalability of the network are also severely constrained by a singular controller. Consequently, multiple controllers

are suggested; but they increase the complexity of networks. At last highlights potential research issues in multiple controllers of SDN such as coordination between controllers, load balancing among controllers, etc.

**A. U. Rehman et al.** [67] Fault tolerance, which guarantees high availability and reliability in the system, is a crucial element of network resilience. Furthermore, they highlight fault tolerance issues in SDN due to a single controller. Thus, SDN controllers still exist with fault tolerance issue which is further categorized into controller reliability, consistency, controller placement, and controller assignment (balance of controllers). Moreover, scalability, availability, and data consistency are still an area of research in SDN for multiple controllers.

**S. Asadollahi et al.** [69] computer networks claim that traditional networks are replaced by SDN because they overwhelmed the limitations of traditional networks. In SDN, the Ryu controller is used for scalability purposes by using the Mininet simulator.

**Y. Chen et al.** [70] multiple controllers in SDN have a critical issue regarding load balancing. Imbalance load may be causing some controllers to be overloaded, and some controllers are underutilized in the network; because a load of each controller is changed w.r.t. time. One of the biggest issues that occur among multiple controllers is the distribution of load between controllers.

**Y. Zhou et al.** [71] Software Defined Network develops a centralized controller which manages the entire network control, but it can suffer from scalability and reliability issues in the control plane. Instead of attempting to balance the load across numerous controllers, the proposed load balancing scheme is built on the switch group.

**J. Yu et al.** [72] Due to a single controller, SDN currently has scalability and availability problems. To fix this problem, implement multiple controllers in SDN. But arises a load balance issue due to the uneven distribution of workload between controllers.

**J. Ansell et al.** [73] Queuing theory is a well-established branch of analysis of the performance changes as network traffic in the network. To study how network



performance will be affected by changing network traffic in the network. So, this concept may be used for load balancing in SDN.

**G. Nencioni et al.** [74] SDN improves the flexibility and programming of networks. But it also brings challenges like the consistency of network, load balancing, fault tolerance, etc. that need to be explored. The operational and management failures (O&M) have a higher impact on overall network availability in SDN. To reduce the operational and management failures (O&M) to design a proper SDN controller is needed.

**B. Xiong et al.** [75] limitation of the logically centralized controller in SDN affects the network performance. A queuing model was proposed for approximating the future performance of SDN controllers.

**D. Chourishi et al.** [77] The development of Software Defined Networking (SDN) is still in its infancy. With SDN, load balancing at the control layer is a major problem.

**K. Benzekki et al.** [78] compare the characteristics of Classical architecture with Software Defined Network architecture. The SDN offers programmability, centralized control, network flexibility, easy implementation and configuration, and enhanced network management as compared to classical networks. At last addressed SDN challenges that are faced such as scalability, reliability, dependability, resiliency, high availability, etc.

**R. K. Das et al.** [81] A single controller's performance is necessary for a network to function, although a solitary point of failure is always a possibility. The performance of a network declines as network traffic increases, which is not what is wanted. Therefore, the system's limited fault tolerance and dependability capabilities are unacceptable.

**T. Hu et al.** [83] SDN provides flexibility in network management. But a single controller does not meet the demand of the network as increasing the network coverage area. To overcome this situation by using multiple controllers in SDN for large network areas. But different four aspects that must be considered while implementing multiple controllers in SDN are scalability, reliability, consistency,



and load balancing. In the future, must be a focus on these challenges as a research area in SDN.

**S. Muhizi et al.** [85] to the evaluation of SDN performance by using a queuing model that monitors. What impact will network traffic have on performance? Furthermore, extend this analysis for multiple controllers in SDN.

**T. Issa et al.** [86] the use of a single controller in SDN creates more network failure points and compromises the control plane's scalability, availability, and the speed. To manage the load on controllers and enhance network performance, multiple controllers are required to do away with a single point of failure.

**H. Yu et al.** [87] distributed controllers are used in SDN to resolve the issue of scalability and load balancing. This paper says while managing load balancing in SDN, then minimizing packet processing delay.

**J. Cui et al.** [88] single centralized controller suffers scalability and reliability in SDN. But in distributed controllers, the key limitation is the counter of uneven distribution of load between controllers. More difficult to handle the variation of load in network traffic. Therefore, multiple controllers introduce a new challenge in the control plane to rebalance a load of controllers when the uneven distribution of load occurs.

**W. H. F. Aly et al.** [89] an important aspect of resilience is fault tolerance which ensures the availability and reliability of the network is high. Both fault tolerance and load balancing are interrelated issues. Moreover, it manages the load between controllers by using performance metrics of the network.

**O. Akanbi et al.** [90] the efficiency of the network and the scalability of the control plane depend heavily on the task distribution among distributed controllers. To address this issue there is two main concern of network controller consider flow setup latency and switch assignment to distribute workload across multiple controllers.

**J. Xu et al.** [91] various strategies are proposed to resolve these issues like load balancing, performance, and robustness of SDN controllers. While multi-controller

deployment in SDN by using queue system; then what is an impact on cost deployment of the controller in network and how it is minimized in future.

**A. Mahjoubi** *et al.* [92] in SDN single controller suffers serious problems like scalability, availability, and central point of failure. Distributed controllers are employed as a solution to these problems. But still, they have to deal with fault tolerance and load balancing challenges among controllers.

**A. Mondal** *et al.* [94] in the future, there is a need for an analysis model to ensure quality-of-service with minimum packet drop in the system, reducing waiting time in a system while using TCAM memory in the system.

**M. Escheikh** *et al.* [95] multiple controllers are used in SDN which suffers several issues as the bottleneck, availability, synchronization, and an unbalanced traffic load while scaling the network size.

**L. Mamushiane** *et al.* [103] by decoupling the both planes in SDN poses several challenges regarding scalability, fault tolerance, load balancing, and network performance. To resolve these issues, deploy multiple controller networks. But in the future, try to integrate both load balancing and fault tolerance into a solution.

**M. K. Faraj** *et al.* [104] load balancing strategy is required when congestion and overloading problem has occurred in the network. In the future, the queue length is utilized for load balancing to reduce congestion in the network; while using multiple controllers rather than a single controller.

**A. Mondal** *et al.* [105] This study suggested a Markov chain-based SDN analytical model for analysing the efficiency of packet flow using OpenFlow. There were a lot of network packet drops; either a table-miss entry happened or there was a long delay before an output action was defined. In the future, it can be expanded with a queue system that helps to shorten the packet flow's latency.

**M. Hamdan** *et al.* [106] in SDN, load balancing is a method used to boost network performance. The main goals of load balancing are a minimum reaction time, effective resource use, maximum system throughput, and avoiding bottleneck issues. During the load balancing, several issues are rising like controller failure, migration of switch(es), managing the load of the controller, resources allocations,

synchronization, controller placement, and so on. These open issues provide further research direction in SDN.

**S. Rowshanrad** *et al.* [109] a controller can communicate with forwarding elements via the OpenFlow which provides flow statistics of a network to the controller for monitoring the network system. In the future, flow statistics information is combined with queueing techniques to get optimize performance in a network.

**G. Huang** *et al.* [110] contrast the software defined network's architecture with that of a conventional network. In comparison to traditional networks, SDN has a number of benefits, including programmability, agility, flexibility, and centralised control. This study adopted a proactive strategy based on the usage of a flow table as opposed to a reactive one. The likelihood of flow entries matching is thus at its highest. Utilize an analytical model in the future to enhance the performance-related factors.

**Y. Zhang** *et al.* [66] based on their interactions, the two kinds of SDN multiple controller architecture are centralised architecture and distributed architecture. The initial SDN architecture, which only implemented one controller, is referred to as centralised architecture. Distributed architecture departs from the original trend of SDN by offering a distributed control plane through the use of several controllers. With distributed architecture, controller scalability can be increased. This kind of design allows for more than one controller to make up the control plane. These numerous controllers are responsible for overseeing various network administrative domains and exchanging local data with neighbouring domains to improve the execution of all global rules and regulations. Mostly concentrate on distributed architecture, which may be divided into two types: horizontal and hierarchical.

- **Horizontal Architecture:** All controller is assigned to a different domain in a horizontal design, commonly referred to as a flat model. East-West interfaces are used by controllers to communicate with each other while managing their own networks independently and with equal state. In a horizontal design, every controller must be on the same level. The usual examples of horizontal architecture include Onix [113], HyperFlow [117], and FlowVisor [114].
- **Hierarchical Architecture:** A hierarchical architecture for managing or controlling controllers. Through a hierarchical architecture, all controllers

are separated into root or master controllers and local controllers based on their function. Local controllers are accountable for the network state within their local domain and are situated reasonably close to switches. The entire network's information must be maintained by root controllers. Local controllers have to communicate with root controllers beforehand handling inter-domain activities, much like the client and server method. IRIS [119] and Kandoo [120] are two examples of distributed SDN controller initiatives utilising a hierarchical paradigm.

When a network experiences a fault, it can still function normally. This is referred to as fault tolerance. Fault tolerance is a preventative method of enhancing controller reliability that guarantees a secure functioning and high performance of networks. Similar to traditional networks, one of the main objectives of controller design is to accomplish fault-tolerant communication [66].

However, unlike conventional networks, SDN's failure tolerance is dependent on the robustness of both the distributed control plane and the data plane. A review of the literature on SDN controllers and their features, as well as the fault-tolerant approach used by the present multiple controller systems, are summarised in Table 5. The effects of controller errors can be greatly reduced by using several controllers.

The placement of controllers must be carefully considered in order to achieve fault tolerance. Most recent studies focus on passive or active replication strategies to increase fault tolerance [66].

- **Passive Replication:** With each switch, only one controller can initiate communication (primary controller). A backup controller will be chosen to take over control of the network when the main controller experiences a failure situation. Primary-backup replication is another term for this strategy.
- **Active Replication:** It is yet another replication technique that is extensively used. Switches can be linked to multiple controllers simultaneously using this technique, allowing the others to continue operating the switches without switching even if one controller fails.

**Table 5: Literature Review on SDN Controller's**

<b>Name</b>	HyperFlow [112]	Onix [113]	FlowVisitor [114]	DISCO [115]	ONOS [116]	Hydra [117]	ElastiCon [118]	IRIS [119]	Kandoo[120]	SMarRLight [121]	OpenDaylight [122]
<b>Language</b>	C++	C++, Java, Python	C	Java	Java	Java	Java	Java	C, C++, Python	C++, Java	Java
<b>Controller Platform</b>	NOX	--	OpenFlow Controller as proxy	Floodlight	ONOS	--	Floodlight	Floodlight	--	Floodlight	OpenDaylight
<b>Open Source</b>	Yes	No	Yes	No	Yes	No	No	Yes	Yes	No	Yes
<b>Failure Type</b>	Controller Failure	Link, Switch and Onix Instances Failure	--	--	ONOS Instances Failure	Controller Failure	Controller Failure	Controller Failure	--	Controller Failure	--
<b>Controller Architecture</b>	Horizontal	Horizontal	Horizontal	Horizontal	Horizontal	--	--	Hierarchical (Root and Local Controller)	Hierarchical (Without global information)	--	--
<b>Controller Load Balancing</b>	×	√	×	×	×	×	√	√	×	×	×
<b>Solution</b>	Nearly Controllers serve as a hot standby	Active Replication	--	--	Redundant Instances	Replication of Controllers Configuration	Dynamic Controller Migration	Controller Switching	--	Replicated shared database for recovery	--
<b>License</b>	--	Commercial	--	--	--	--	--	--	--	--	EPL v1.0
<b>OpenFlow Version</b>	v1.0	v1.0	--	v1.1	v1.0, v1.2, v1.3	--	v1.0	v1.0, v1.2, v1.3	v1.0	v1.0, v1.2, v1.3	v1.0, v1.2, v1.3
<b>Northbound APIs</b>	REST API	REST API	--	REST API	REST API	--	REST API	REST API	--	REST API	REST API

### 3.3.1 SDN Protocols

The SDN paradigm makes use of southbound APIs to facilitate communication between the network plane and the forwarding plane. The SDN controller has effective network management, allowing it to adapt the network dynamically to meet demand and needs as they arise. The OpenFlow protocol is the most widely used communication protocol between the SDN controller and the network devices; Other protocols that can be used as southbound interfaces in SDN include Network Configuration Protocol (NetConf), which was developed by the Internet Engineering Task Force (IETF) and allows users to install, modify, erase, or remove network device configuration by using an Extensible Markup Language. (XML). It provides a basic set of operations to edit and query configuration on forwarding devices. The map and encapsulate functions are supported by the Locator/Identifier Separation Protocol (LISP), which is created by the Internet Engineering Task Force LISP Working Group. LISP is in charge of determining the relationship between EID (End Point Identifiers) and RLOC (Route Locators). The entire process is completely unseen (invisible) from the end host of the internet. OVSDB (Open vSwitch Database) is a southbound API that allows to managing and manipulation of the configuration of switches that support OVSDB through JSON RPC (JavaScript Object Notation Remote Procedure Call) calls. Thus, it is called a management protocol in an SDN environment. However, the most well-known SDN standard for southbound APIs is OpenFlow.

### 3.3.2 OpenFlow Protocol

The OpenFlow Protocol has been commonly associated with SDN since 2011. The SDN Controller establishes a secure communication channel with forwarding devices by using the OpenFlow protocol. SDN network enables switches from different vendors to be managed remotely using OpenFlow protocol. The version of the OpenFlow protocol should be matched between the controller and network devices when they make a communication connection.

**W. Braun** *et al.* [8] the current network's lack of programming should be addressed. By utilising Software Defined Networking, which dynamically adds new capabilities to the network for a set of applications, this issue can be resolved. The

OpenFlow Protocol is one of the most widely used and recognised southbound APIs in SDN. It was originally suggested by Stanford University. Currently, the Open Networking Foundation (ONF) has standardised it. For this purpose, the OpenFlow protocol supports numerous kinds of messages which define how a controller coordinates these forwarding devices in a network. All messages fall into one of three categories: controller-to-switch, asynchronous, or symmetric. The processing speed of a packet affects how long it spends in the controller overall. Additionally, one unresolved problem in SDN is determining a controller's packet drop probability.

**L. Shif** *et al.* [14] address the top two challenges such as security and scalability. These challenges can be improved by SD-VPN (Software Defined Virtual Private Network). The SDN controller pushes the flow rules in each virtual private network (VPN) which is related to OpenvSwitch through the OpenFlow protocol.

**C. M. Duran** *et al.* [19] utilize OpenFlow select and fast-failover group to set up flows in switches of a network and system response in failure. This will improve the performance of the network.

**W. Li** *et al.* [20] a leading reference to the SDN OpenFlow. It has mainly three components OpenFlow switch, OpenFlow controller, and OpenFlow flow channel. When a new packet comes into the network, the switch matches its header field against the flow entries of a table. The appropriate action is carried out if the packet's header field matches; otherwise, switch forward the packet to a controller. The controller inserts or alters flow entries in switches. After that switch can easily forward packets to the desired destination. So, try to develop SDN with OpenFlow in the future.

### 3.4 Observations and Research Gaps

Software Defined Network is a new emerging paradigm of networking that offers flexibility and fast innovation compared to the traditional network. In SDN, decoupling the control plane from the forwarding plane provides a programmability facility to configure the network elements. But a single controller has suffered

restrictions on the performance and scalability of the network. A criticism in SDN is a single point of failure of a controller that reduces overall network performance, and availability, and at last, collapses the entire network. When multiple controllers are used in the network then increases the complexity of networks and arise potential research issues in SDN such as coordination between controllers, fault tolerance, load balancing among controllers, controller placement, etc.

Several challenges/issues are existing in Software Defined Networks when dealing with multiple controllers to solve the above problem [24,64,66] are: -

- Coordination between the multiple controller and switches: - When Software Defined Networks are deployed in a wider network; it consists of several domains that are managed with multiple controllers. In the data plane, different operators implement different policies in forwarding devices that may arise conflict between these policies; a question is a rise how to manage the coordination between these policies and avoid conflict between them. It is a critical issue in SDN.
- When multiple controllers are used in Software Defined Networks, fault tolerance mechanisms are not clearly described: - At present, there is no well-defined fault tolerance mechanism for multiple controllers because, during this process, a critical issue arises regarding the number of controllers and their appropriate locations.
- Controller Placement: In case of multiple controllers arise two critical issues: how to decide on the number of controllers and where in the network they should be placed. In the implementation of SDN multiple controllers, they play a crucial part [66]. But both issues are NP hard problems. So, fault tolerance at control plane is very complicate because as the number of controllers rises then overall cost of network is hike. No universal rule exists for controller placement; So, there must be to find an optimal trade-off between the reliability and latency of the network.
- The dynamic load balancing mechanism for the multiple controllers: - When organising numerous controllers in a large-scale network, it requires concrete information on network traffic to manage the performance of the network in



an optimised way. Thus, network traffic requires a dynamic load balancing mechanism that adapts and adjusts the controller load dynamically. So, an effective load balancing mechanism is required for this purpose.

- Absence of standardization of the east-west bound interface in SDN: - There is an absence of a standardised protocol for the east-west interface between multiple controllers' communications. And no facility for consistency among the heterogeneous controllers; which increases the latency time during the load balancing of controllers.
- Lack of standardization for the northbound interface: - SDN provides an open-source for northbound APIs which enables to development of the network applications. Several types of controllers used different languages for developing northbound interfaces; which increases the complexity between controllers to manage the interfaces in a large network. So, to develop a standardized northbound interface that hides the complexity between controllers.
- Integration of network virtualization and SDN for multiple controllers: - Network virtualization hides the infrastructure underneath, which can change based on how much work is being done and how flexible the resources need to be. Thus, an integration of both paradigms offers an innovative design that has advantages for both SDN and NFV.
- The cost field does not associate with flow entries of a flow table: - When more than one of the flow rules has the same priority. Then conflicts arise between these flow rules. To address this issue, the flow table must include a new cost-related field.

### **3.5 Objectives**

The main goal of the thesis is to design a model for fault tolerance and load balancing in SDN. In a traditional network, it becomes more complicated and tough to handle the widespread adoption of a network because it is complicated to configure the network elements with its predetermined rules and policies. If trying to reconfigure

them to respond to a fault, load, and changes increases the complexity of the network, and sometimes it becomes unmanageable [6,16].

The main reason behind this is a vertical integration exists between both functional components of the traditional networking devices. This makes it very difficult to add new functionality to a network; if try to make any modification or alteration to the control plane then all network elements require to install new firmware and upgrade their hardware devices. So, it can be said that when adding a new feature, it always increases the expensive and difficult to configure the elements due to the change of topology and functionality of a network. To overcome this situation, restructuring the infrastructure of the traditional networks whose name is Software Defined Network (SDN). Decoupling or delinking the control plane from the data plane in SDN improves network efficiency. A controller, however, plays a crucial role in the SDN architecture because it offers a centralized, global perspective of the network. Additionally, it raises the network's maximum possibility of failure. As a consequence, the whole network operation becomes halted. For this reason, a single controller may not be feasible for a network. This revealed that fault tolerance and recovery from failure are the major challenges faced in the SDN now. To overcome these challenges, the researchers are devoting more efforts toward the SDN controller for the great availability, scalability, performance, and reliability of services.

Based on the study, the main objectives of the thesis would be:

1. To design a model for proactive fault tolerance in SDN to reduce single point failure.
2. To design an adaptive algorithm for load balancing in SDN controllers.
3. Validation of proposed fault tolerance and load balancing algorithm for SDN.

### **3.6 Summary**

A brand-new networking paradigm called the “Software Defined Network” enables faster innovation and flexibility than the “traditional network.” However, a single

controller has hampered the network's performance and scalability. A single point of failure for a controller that lowers overall network performance and availability before ultimately collapsing the entire network is a critique of SDN. The usage of many controllers in a network increases network complexity and raises possible SDN research questions on controller coordination, fault tolerance, load balancing, controller placement, and other related topics.

# **Chapter 4**

## **Significance of Controller in Software Defined Networks**

# **4 CHAPTER**

## **Significance of Controller in Software Defined Networks**

### **4.1 Introduction**

By segregating the control plane from the forwarding plane and allowing for the programmability of network components, software defined networks reduce the network's complexity. The SDN controller then makes it easy and straightforward to modify network policies because it offers a logically centralised perspective of a complete network. Also, it provides facilities such as flexibility, centralized view control, decrease complexity as well as a decrease in the cost of a network system. The network innovation provides a position to SDN as the forthcoming of networking. The novel paradigm of SDN is more flexible than traditional networks by separating vertical integration between both functional components.

When a user quickly changes their demand for resources then the Software Defined Network is quickly satisfied their needs as soon as possible. Till now, SDN has undertaken continuous development in academia and industry area. The controller acts as a critical component in Software Defined Network because it provides a logically centralized view of a whole network. Therefore, it increases the maximum chances of failure in the network due to a single controller [59] that has faced many issues regarding scalability, fault tolerance, and recovery from these failures in the network. To overcome these issues, the researchers can make more efforts toward

the SDN controller for increasing its performance, scalability, reliability, resiliency, and availability of the controller [6,15-28,52-67].

It is up to the controller to determine how to manage network traffic in the data plane. When a new packet enters a network then the data plane receives forwarding orders from the SDN controller. After receiving these instructions then forwarding elements are updating their flow tables according to these instructions that are provided by a controller. Thus, SDN networks are capable for communicate and manage the forwarding elements which are supplied by the various vendors [16,59]. Because SDN transforms the communication network into a programmable network that enables the service provider or network operator to update the network faster/sooner and decrease the capital and operational expenditure (CapEx and OpEx).

## **4.2 Related Work**

Software Defined Network provides a novel paradigm for the network that separates the two planes. Moreover, SDN has not been a talent to recover from a failure automatically. So, to design a mechanism that offers coordination between controller and switches like a fast failover method. Three factors have a great impact on the recovery process network changes discovery, path computation, and network updating. But, recovering from multiple link failures remains an open issue. It can be considered a forthcoming research direction because lack of resilience failure and significant elements that have an impact on network performance include the increase in convergence time following a failure. Thus, SDN has brought a lot of opportunities for innovation in the networking field. But still, it has faced several challenges like fault tolerance, recovery, a resilience that degrade the reliability, scalability, and availability of networks [16].

The controller and switches can communicate with one another through the OpenFlow interface. The dedicated load balancing in SDN becomes expensive due to a single controller increasing congestion in a network. Thus, TCAM memory is preferable to hold flow tables because it provides flexibility and efficiency in terms

of the matching capabilities but, it is very expensive and small in size. So, TCAM memory is insufficient to hold a large number of flow entries in a flow table simultaneously [6,16,27]. To manage the OpenFlow Protocol has the facility to use wildcard rules in the flow table. Another perspective is to occur a fault between both controller and switch(es). If any fault occurs between them, then their switches lost their connection with a controller or if a controller has broken down then underlying switches become out of the control. As a consequence, the total network becomes halts [18,24, 52-58]. To overcome this situation by using multiple controllers to eliminate the risk of a central point of failure in SDN. So, the concept of multiple controllers in SDN considers the future research direction/area.

To enhance the resiliency of the network by using multiple controllers. When a primary controller is a breakdown then the backup controller manages the entire network. But these changes can be arising inconsistency in the network. To overcome this situation by managing all modifications in network topology simultaneously. There is one critical problem that can occur regarding the appropriate number of controllers to use as well as where they should be placed within the controllers. For this reason, to maintain a trade-off between these metrics like latency, reliability, and load balancing [66].

### **4.3 Problem Formulation**

In the SDN paradigm, a centralised controller oversees all network components in a network. For two primary reasons, relying on a controller is not feasible. The first is that the controller is constantly at risk of becoming a point of failure (SPOF) in the network [59]. Second, it halts an entire network operation which has an adverse effect on the network performance. Sometimes, a bottleneck situation can be arising in the network when a controller handles a large number of switches and need to send instructions to these forwarding elements/devices on how to control network traffic on demand. As a result, it increases the latency time and reduces the throughput of a network [21,58,59].

Thus, Software Defined Network improves efficiency, programmability, and utilization of the network. Due to a single/solitary controller, it restricted the scalability and reliability of the SDN paradigm. For this purpose, focusing on the multiple controllers in SDN increases the scalability and availability of a network. An eliminate the threat of a SPOF in the network for future development [6,66]. So, it must provide the facility of fault tolerance in the SDN network. There is only 4% of research effort devoted/contributed to fault tolerance and 96% in other fields of SDN [16].

A mechanism known as fault tolerance that empowers the system to continue its functionality or operations even if a failure occurs or is present in its components. In a communication network, fault tolerance has been widely used because it provides a mechanism for how to recover from failure when it has happened in the system [15,16]. In SDN, fault tolerance of the control plane concerned more attention; because it always provides a logical view of an entire network. That is why a controller acts as a critical component in SDN. If any fault occurs in the controller, then complete network operation becomes halted. So, a fault tolerance mechanism is required to manage these faults.

In SDN, the controller is a crucial component; so maximum efforts have been dedicated to the SDN controller for achieving high performance, scalability, reliability, and availability of network services to the user. In a distributed environment, controllers have to address various challenges that give particular attention to the problems of consistency, load distribution, fault tolerance, latency time, etc. SDN centralized control of a controller poses a threat to the central point of failure in the network and it is not able to recover from failure itself. So, the main concern is to achieve maximum resilience, availability, and scalability of multiple controllers cost-effectively [6,16,64,66].

When a single controller installs a lot of flow rules, the network's overheads are greatly increased. A trade-off between load balancing and network delay must be taken into account in order to avoid this predicament. Due to the inefficiency of a single controller in a large-scale network and the fact that increased network traffic lowers a network's consistency and performance, load balancing is therefore



required. As a consequence, increases the loss of packets in a network. At last, a moment has occurred when an entire network collapses due to the failure of a controller. Thus, a fault tolerance mechanism is required to avoid or eliminate a single point of failure (SPOF) in SDN to use multiple controllers.

#### **4.4 Why the Controller is an Essential or Crucial Component to SDN**

The SDN has opened up numerous opportunities for networking industry growth. However, the controller plays a crucial role in SDN because it offers a comprehensive view of the complete network. That is why it is called the “*Brain of Network.*” It acts as a bridge connecting the application plane and data plane. The controller can configure or reconfigure the network devices by dynamically customizing their policies; thus, a developer is free from the need to maintain information about the low-level detail of data distribution among the forwarding devices while defining their network policies [51-68]. Therefore, the controller not only manages the network but also resolves its problem. So, maximum efforts have been devoted to the SDN controller for achieving high performance, with faster scalability, more reliability, and ready availability of services; and provide an advantage over a centralized control setup risk of failure in the network and its limitation to recover automatically from a failure. So, our main concern is to achieve maximum resilience and scalability of SDN by cost-effectively using multiple controllers [2, 6-9,15-66]. In a distributed environment, the controller has to deal with many different kinds of problems, such as latency time, tolerance to faults, balance of load, consistency, and many other issues [6–21]. The controller considers a critical component in SDN for the following reasons:

- A logically centralised view of the network is provided by the controller.
- It can control the network traffic.
- It is the controller's duty to update and maintain network topology data.

- The controller has the authority to install the flow rules in forwarding elements either using a reactive or proactive manner.
- Through the programmability, the controller decreases the complexity network.
- The controller has the power to control the functioning of forwarding elements.
- When the controller handles more traffic as compared to its available capacity. Then it increases the latency of the network and degrades its performance of a network.

The SDN controller is also responsible to maintain the entire network topology information. For this purpose, the controller generates the LLDP (Link Layer Discovery Protocol) packet to determine (or discover) the topology information about each switch, port, and link in the network as shown in Figure 4.1.

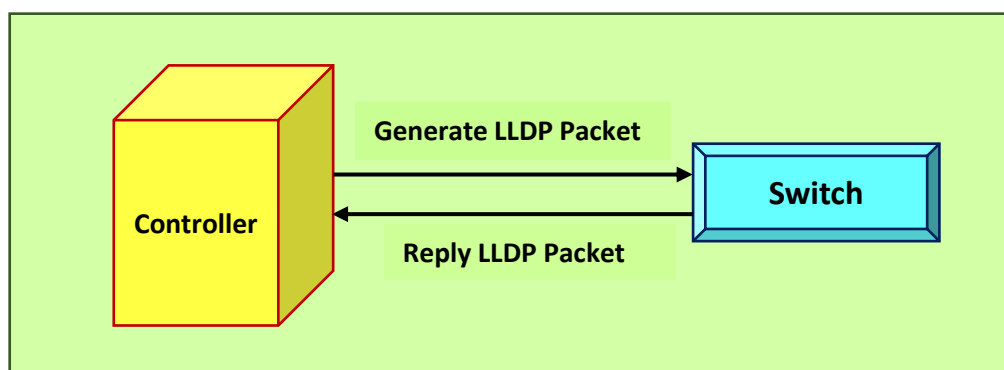


Figure 4.1: Use of LLDP Packet between the Controller and Switch in SDN

## 4.5 Single Controller v/s Multiple Controllers

An SDN offers flexibility, and a centralized view of the network thus helping in decreasing the complexity and cost of the network. Because a controller has the skill to decide on how to control network traffic in the forwarding plane through the flow rules [7,16]. It acts as a critical component in SDN. A single controller may not be feasible in SDN because it has a higher impact on failure in the network due to its centralized view. If a failure has occurred, then the entire network becomes collapses. This situation is not desirable in a communication network. To overcome

this problem, a fault tolerance mechanism is required that eliminates the possibility of an SDN single point of failure [59]. To overcome this situation, researchers can make more efforts toward the SDN controller for increasing the performance, scalability, reliability, and availability of the controller [6-9,15-23,58-66]. For this purpose, multiple controllers are used in Software Defined Network to lessen a central point of failure in SDN as shown in Figure 4.2.

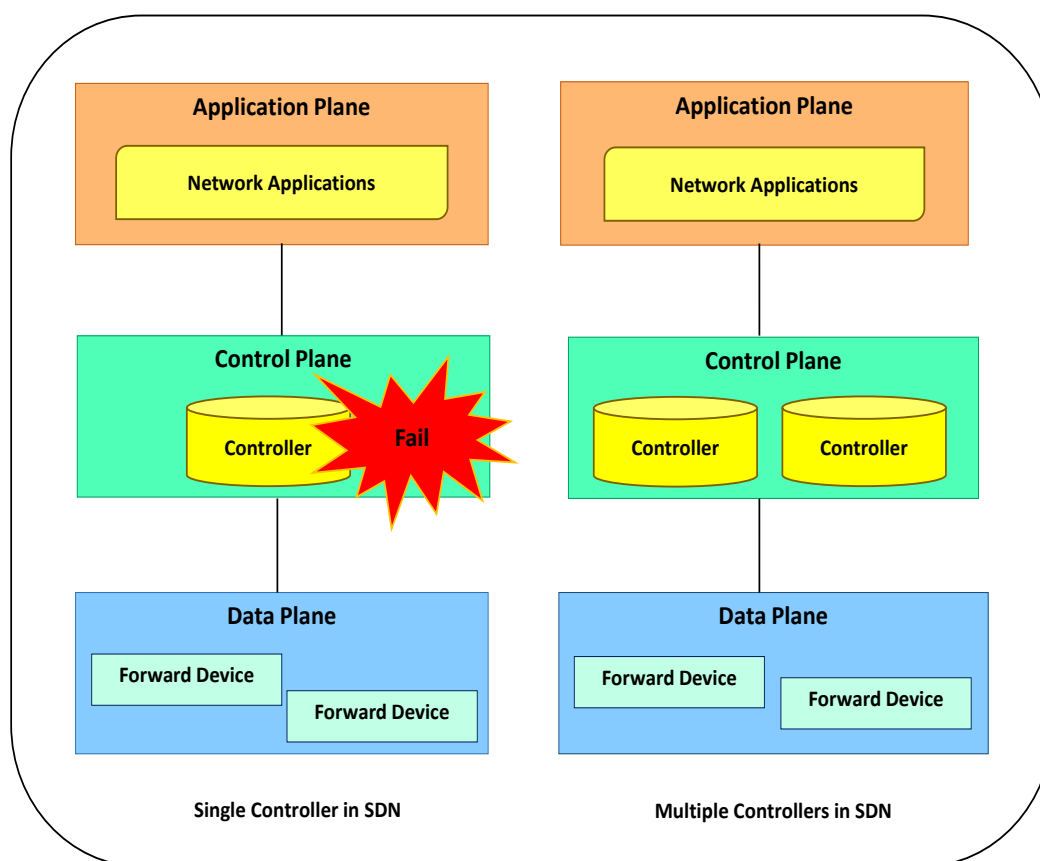


Figure 4.2: Single Controller v/s Multiple Controllers in SDN

## 4.6 Type of Multiple Controller’s in SDN

The single point of failure in SDN can be eliminated by using multiple controllers in the network. The OpenFlow specification support multiple controllers’ environment in SDN, in which controller can perform role among the following three types (as shown in Figure 4.3).

1. **Equal Role:** All controllers have the privilege to configure the switches in the network with full control to update or alter the flow rules. When a switch

sends a PACKET-IN message to all the controllers and processes messages like PACKET-OUT, FLOW-MOD, etc. from all controllers.

2. **Master Role:** The master controller has responsibility for managing or handling the switches in the network. Thus, switches will send only control messages to the master controller.
3. **Slave Role:** The slave controller act as a backup role for the master controller. It can only receive HELLO and ECHO messages in the network. But they cannot send and receive control messages.

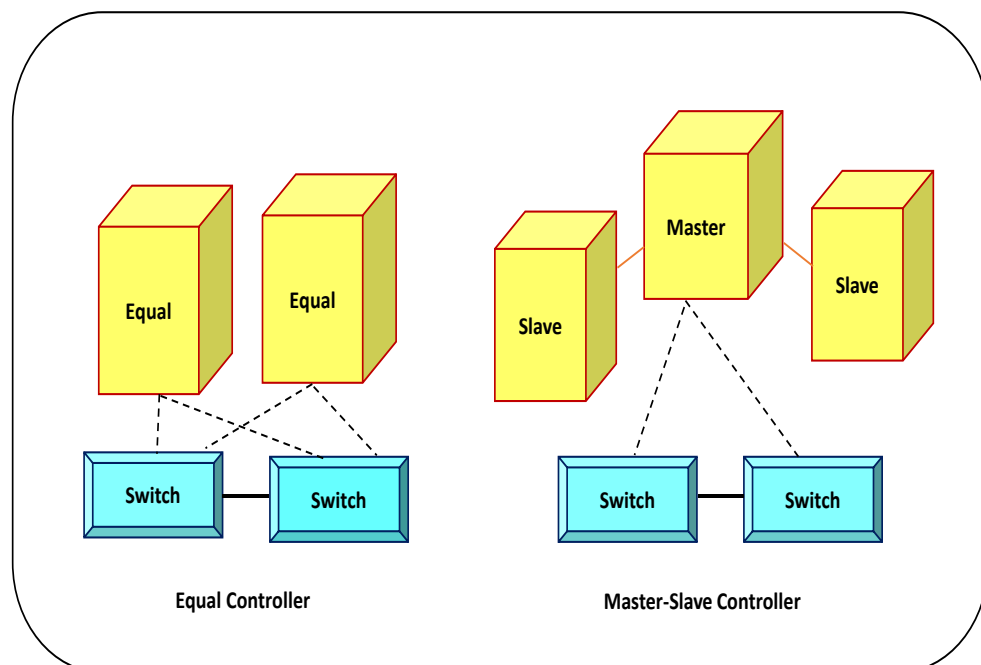


Figure 4.3: Various Role of the Multiple Controller in SDN

Every switch in the network can only have one master controller. However, it can have a number of equal or slave controllers. Even if the master controller fails, the network will still be more reliable because there are so many controllers [59]. Then any other controller sends a ROLE\_REQUEST message as the ROLE\_MASTER to switch for changing their state or role. On receiving this message, the switch sends back the ROLE\_REPLY message to the controller and other controllers sends the role as SLAVE in the network. Thus, the switch will communicate only with the master controller [6, 16, 59-67]. The behavior of multiple controllers is summarized

in Figure 4.4 and Figure 4.5 shows the pseudo-code for SDN's single point of failure reduction.

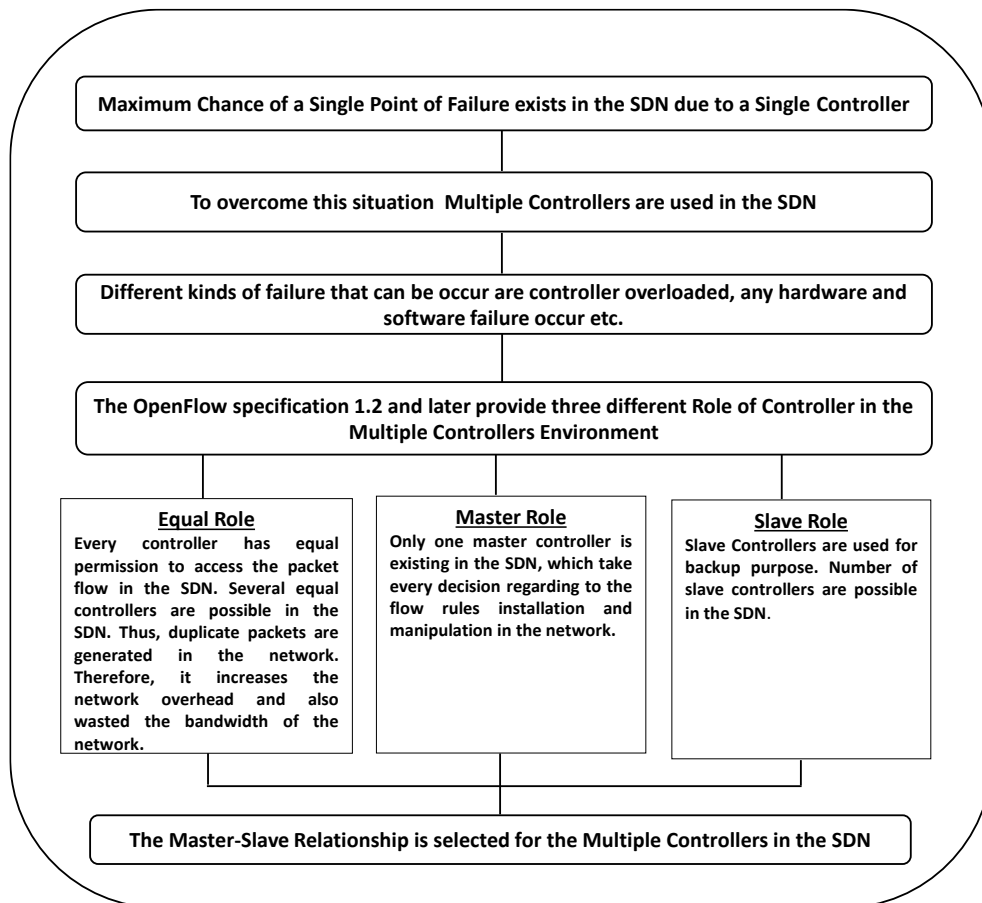


Figure 4.4: Behavior of Multiple Controller in SDN

***/\* Pseudo Code to Reduce a Single Point of Failure in SDN \*/***

**Step 1:**

Start the ryu-manager with ofp-tcp-listen-port 6653 as a single controller;

**Step 2:**

```

if (Single Controller == "Fail")
{
    /* When a single point of failure (SPOF) occurs in SDN */
    Collapse the Entire Network;
}
    
```

```
else
{
    /* To eliminate a single point of failure by using Multiple Controllers in
        SDN*/
    /* The OpenFlow 1.2 and its later specification provide three different
        roles of controller's*/
    switch (ROLE)
    {
        case ROLE == "EQUAL":
            

- Every controller has permission to access the packet flow in the network.
- Several equal controllers exist in the network.
- Duplicate packets are generated and marked as (DUP!).


            break;
        case ROLE == "MASTER":
            

- Only one master controller exists in the network.
- All decision regarding installation and manipulations of flow rules are taken by Master Controller.


            if (MASTER == "Fail")
            {
                Any slave controller sends Role-Request message to switch;
                if (ROLE_REQUEST == "ROLE_MASTER")
                {
                    Switch send reply to Controller;
                    ROLE_REPLY == "MASTER";
                }
            }
            else
            {
                Other controllers send role as a slave controller;
```

```
        }
    }
    break;
case ROLE == "SLAVE":
    • Slave controller used as backup purpose.
    • Number of slave controllers exist in the network.
    break;
}
}
```

**Output:**

Therefore, multiple controllers are used to reduce a single point of failure in SDN.

Figure 4.5: Pseudo Code to Reduce Single Point of Failure in SDN

## 4.7 Simulation and Result

Various controllers are available like Floodlight, Ryu, Pox, ONOS, OpenDayLight, etc. In the proposed objective the Ryu controller is elected; because it is open source and designed to magnify the agility in a network. It is completely based on the Python language. Therefore, it is too easy to manage and adapt network traffic to the network. Ryu controller provides a component-based framework for Software Defined Networks. Through precisely defined APIs, the Ryu controller offers software component capability. Developers are capable of developing new network management and control apps with ease. The component-based approach offers the facility to the organizations by customizing deployments to satisfy their specific demand. As a result, the programmers can quickly and skilfully put their applications into use by modifying the already-existing components to fit their requirements.

Ryu uses different protocols for controlling network devices as OVSDB, BGP, and OpenFlow. The Ryu fully supports OpenFlow v1.0, v1.2, v1.3, v1.4, v1.5, and Nicira Extensions. Ryu means “flow” in Japanese, and it is pronounced, “ree-yooh.” Ryu

controller is highly used to develop and support research activities. The Ryu controller support southbound interfaces are OpenFlow and northbound interfaces are REST APIs [69,98,99].

#### 4.7.1 Mininet Simulator

Mininet is a network emulator that runs on a virtual machine. It can make a realistic virtual network for simulation; because the switch, kernel, and application code are all executed on the same terminal. Mininet offers both environments for simulation command-line interface (CLI) and an application programming interface (API). It is lightweight OS virtualization to achieve scalability. Mininet simulator also provides the following features are [102]:

- Open Source
- Fast
- Possible to create custom topologies
- Can run real programs
- Many OpenFlow features are built-in
- Programmable OpenFlow switches
- Easy to use

The Mininet simulator also provides the facility to build a custom topology utilizing the python API with a few lines of code that import the required python libraries, define topology class, and then create topology class objects. After saving the filename with the .py extension.

To simulate or experimental setup required the following software tools and programming language are used on the experimental platform: Ubuntu 18.04 Desktop as an Operating System, Mininet 2.3.0d5 as a Test Bed, Ryu SDN Controller use as a Remote Controller, Python 2.7.17 version. A laptop with a 64-bit operating system, an x64-based processor, the DESKTOP-3CK61R7, with an Intel(R) Core (TM) i5-8250U CPU @ 1.60GHz 1.80GHz as a CPU, and 4 GB of RAM are all included in the hardware environment. Table 6 details every simulation parameter in depth. After these tools have been installed successfully, simulations of the various controller roles in SDN are run.



Table 6: Simulation Parameters of Setup

Parameters of Simulation Setup	
Operating System	Ubuntu 18.04.2 LTS (Long Term Support version of Ubuntu)
System Specification	x64 Intel(R) Core (TM) i5-8250U CPU
Simulator/Test Bed	Mininet 2.3.0d5
SDN Controller	Ryu Controller (ryu-manger 4.32)
Support of OpenFlow	OpenFlow v1.3
Switch	Openvswitch (2.9.5)
DB Schema	7.15.1
Programming Language	Python 2.7.17
Packet Size (byte)	64
By default, Port No of Controller	6653
IP Address of Remote Controller	127.0.0.1
Bandwidth	10 Mbit/sec
Delay	2ms

During simulation, to analyze when a single controller fails then the whole network becomes unreachable. Therefore, packet loss occurs in the network shown in Figure 4.6 (a to d). To overcome this situation, by using multiple controllers in SDN (either equal controller or master-slave controller). In the case of Equal controller generated duplicate packets in a network as depicted in Figure 4.7 (a to d). However, as can be seen from Figure 4.8 (a to e), the Master-Slave controller does not introduce duplicate packets.

```

dell@dell-Inspiron-3576:~/Desktop/SDN$ ryu-manager --ofp-tcp-listen-port 6653 flow_timeout.py
loading app flow_timeout.py
loading app ryu.controller.ofp_handler
instantiating app flow_timeout.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 2 00:00:00:00:00:02 33:33:ff:00:00:02 1
packet in 1 00:00:00:00:00:02 33:33:ff:00:00:02 2
packet in 2 aa:9b:e3:47:80:bf 33:33:ff:47:80:bf 2
packet in 1 ae:7b:ac:98:c8:d0 33:33:00:00:00:16 2
packet in 1 00:00:00:00:00:01 33:33:ff:00:00:01 1
packet in 2 aa:9b:e3:47:80:bf 33:33:00:00:00:16 2
packet in 2 00:00:00:00:00:01 33:33:ff:00:00:01 2
packet in 1 00:00:00:00:00:01 33:33:00:00:00:16 1
packet in 2 00:00:00:00:00:01 33:33:00:00:00:16 2
packet in 1 ae:7b:ac:98:c8:d0 33:33:ff:98:c8:d0 2
packet in 2 00:00:00:00:00:02 33:33:00:00:00:16 1
packet in 1 00:00:00:00:00:02 33:33:00:00:00:16 2

```

Figure 4.6 (a): Run a SDN Controller in a Terminal

```

dell@dell-Inspiron-3576:~$ sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocol=OpenFlow13 --topo=linear,2
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s2) (s2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet> h1 ping h2
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=12.4 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.514 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=0.096 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=0.071 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=64 time=0.088 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=64 time=0.088 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=64 time=0.090 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=64 time=0.090 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=64 time=0.092 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=64 time=0.091 ms
64 bytes from 10.1.1.2: icmp_seq=11 ttl=64 time=0.103 ms
64 bytes from 10.1.1.2: icmp_seq=12 ttl=64 time=0.091 ms
64 bytes from 10.1.1.2: icmp_seq=13 ttl=64 time=0.096 ms
64 bytes from 10.1.1.2: icmp_seq=14 ttl=64 time=0.096 ms
64 bytes from 10.1.1.2: icmp_seq=15 ttl=64 time=0.099 ms
64 bytes from 10.1.1.2: icmp_seq=16 ttl=64 time=0.078 ms
64 bytes from 10.1.1.2: icmp_seq=17 ttl=64 time=0.092 ms
64 bytes from 10.1.1.2: icmp_seq=18 ttl=64 time=0.128 ms

```

Figure 4.6 (b): Run a topology in a Terminal through Mininet

```

64 bytes from 10.0.0.2: icmp_seq=53 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=54 ttl=64 time=0.085 ms
64 bytes from 10.0.0.2: icmp_seq=55 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=56 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=57 ttl=64 time=0.081 ms
64 bytes from 10.0.0.2: icmp_seq=58 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=59 ttl=64 time=0.086 ms
64 bytes from 10.0.0.2: icmp_seq=60 ttl=64 time=0.095 ms
From 10.0.0.1 icmp_seq=67 Destination Host Unreachable
From 10.0.0.1 icmp_seq=68 Destination Host Unreachable
From 10.0.0.1 icmp_seq=69 Destination Host Unreachable
From 10.0.0.1 icmp_seq=70 Destination Host Unreachable
From 10.0.0.1 icmp_seq=71 Destination Host Unreachable
From 10.0.0.1 icmp_seq=72 Destination Host Unreachable
From 10.0.0.1 icmp_seq=73 Destination Host Unreachable
From 10.0.0.1 icmp_seq=74 Destination Host Unreachable
From 10.0.0.1 icmp_seq=75 Destination Host Unreachable
From 10.0.0.1 icmp_seq=76 Destination Host Unreachable
From 10.0.0.1 icmp_seq=77 Destination Host Unreachable
From 10.0.0.1 icmp_seq=78 Destination Host Unreachable
From 10.0.0.1 icmp_seq=79 Destination Host Unreachable
From 10.0.0.1 icmp_seq=80 Destination Host Unreachable
From 10.0.0.1 icmp_seq=81 Destination Host Unreachable
From 10.0.0.1 icmp_seq=82 Destination Host Unreachable
From 10.0.0.1 icmp_seq=83 Destination Host Unreachable
From 10.0.0.1 icmp_seq=84 Destination Host Unreachable
From 10.0.0.1 icmp_seq=85 Destination Host Unreachable
From 10.0.0.1 icmp_seq=86 Destination Host Unreachable
From 10.0.0.1 icmp_seq=87 Destination Host Unreachable
From 10.0.0.1 icmp_seq=88 Destination Host Unreachable
From 10.0.0.1 icmp_seq=89 Destination Host Unreachable
From 10.0.0.1 icmp_seq=90 Destination Host Unreachable
^C
--- 10.0.0.2 ping statistics ---
91 packets transmitted, 60 received, +24 errors, 34% packet loss, time 92110ms
rtt min/avg/max/mdev = 0.067/0.269/7.230/1.019 ms, pipe 4
mininet>
    
```

Figure 4.6 (c): Host Unreachable after killing the Controller

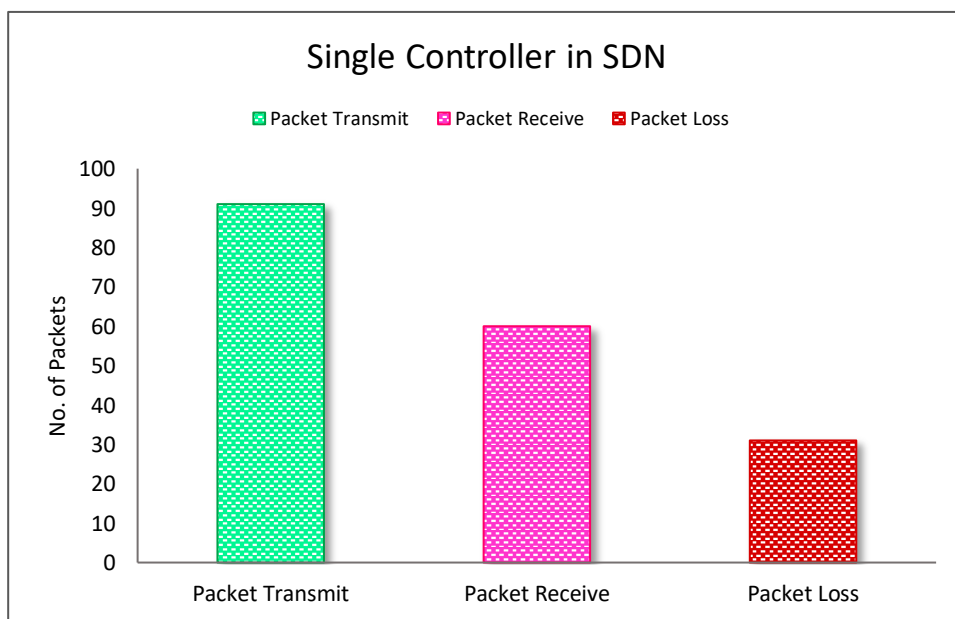


Figure 4.6 (d): Simulation in a Single Controller in SDN

```
dell@dell-Inspiron-3576:~/Desktop/SDN$ ryu-manager --ofp-tcp-listen-port 6653 flow_timeout.py
loading app flow_timeout.py
loading app ryu.controller.ofp_handler
instantiating app flow_timeout.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
█
```

Figure 4.7 (a): Run an Equal Controller in a Terminal at Port Number 6653

```
dell@dell-Inspiron-3576:~/Desktop/SDN$ ryu-manager --ofp-tcp-listen-port 6654 flow_timeout.py
loading app flow_timeout.py
loading app ryu.controller.ofp_handler
instantiating app flow_timeout.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
█
```

Figure 4.7 (b): Run an Equal Controller in a Terminal at Port Number 6654

```
h1 h2
*** Adding switches:
s0 s1 s2
*** Adding links:
(h1, s1) (h2, s2) (s0, s1) (s0, s2)
*** Configuring hosts
h1 h2
*** Starting controller
c0 c1
*** Starting 3 switches
s0 s1 s2 ...
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=56.9 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=58.7 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=60.4 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=61.3 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=61.4 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=61.5 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.694 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.099 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.099 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.107 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.092 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.095 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.104 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.100 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.093 ms
^C
--- 10.0.0.2 ping statistics ---
13 packets transmitted, 13 received, +5 duplicates, 0% packet loss, time 12270ms
rtt min/avg/max/mdev = 0.092/20.124/61.580/28.270 ms
mininet> █
```

Figure 4.7 (c): Duplicate Packets generated during Simulation of Equal Controller

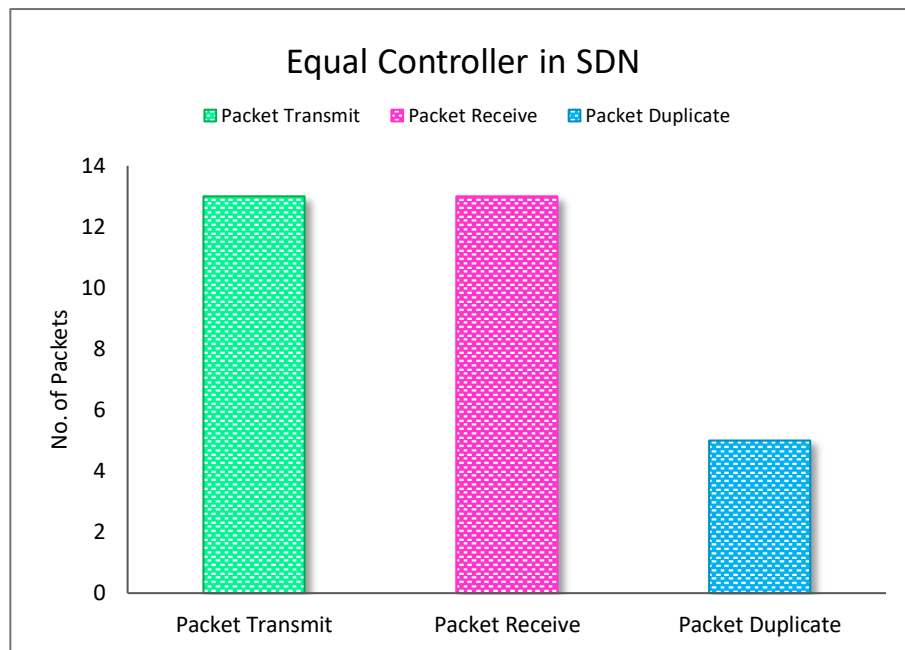


Figure 4.7 (d): Simulation in Equal Controller in SDN

```
dell@dell-Inspiron-3576:~/Desktop/SDN$ ryu-manager --ofp-tcp-listen-port 6653 l3switch_master.py
loading app l3switch_master.py
loading app ryu.controller.ofp_handler
instantiating app l3switch_master.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
This Controller is Master
This Controller is Master
This Controller is Master
packet in 1 00:00:00:00:00:01 33:33:00:00:00:16 1
packet in 187858180115784 00:00:00:00:00:01 33:33:00:00:00:16 1
packet in 2 00:00:00:00:00:01 33:33:00:00:00:16 2
packet in 2 36:c7:a8:ad:33:00 33:33:00:00:00:16 2
packet in 2 00:00:00:00:00:02 33:33:00:00:00:16 1
packet in 187858180115784 00:00:00:00:00:02 33:33:00:00:00:16 2
packet in 1 00:00:00:00:00:02 33:33:00:00:00:16 2
packet in 187858180115784 e6:fa:ae:c8:3c:08 33:33:00:00:00:16 1
packet in 2 e6:fa:ae:c8:3c:08 33:33:00:00:00:16 2
packet in 2 36:c7:a8:ad:33:00 33:33:ff:ad:33:00 2
packet in 1 ee:69:cd:07:9f:d0 33:33:ff:07:9f:d0 2
packet in 187858180115784 e6:fa:ae:c8:3c:08 33:33:ff:c8:3c:08 1
packet in 2 e6:fa:ae:c8:3c:08 33:33:ff:c8:3c:08 2
packet in 2 00:00:00:00:00:02 33:33:ff:00:00:02 1
packet in 187858180115784 00:00:00:00:00:02 33:33:ff:00:00:02 2
packet in 1 00:00:00:00:00:02 33:33:ff:00:00:02 2
packet in 1 00:00:00:00:00:01 33:33:ff:00:00:01 1
packet in 187858180115784 00:00:00:00:00:01 33:33:ff:00:00:01 1
packet in 2 00:00:00:00:00:01 33:33:ff:00:00:01 2
packet in 1 ee:69:cd:07:9f:d0 33:33:00:00:00:16 2
packet in 187858180115784 3e:87:70:d2:59:d2 33:33:ff:d2:59:d2 2
packet in 1 3e:87:70:d2:59:d2 33:33:ff:d2:59:d2 2
packet in 2 36:c7:a8:ad:33:00 33:33:00:00:00:16 2
packet in 2 36:c7:a8:ad:33:00 33:33:00:00:00:02 2
packet in 2 36:c7:a8:ad:33:00 33:33:00:00:00:16 2
packet in 2 36:c7:a8:ad:33:00 33:33:00:00:00:fb 2
packet in 2 36:c7:a8:ad:33:00 33:33:00:00:00:fb 2
packet in 1 ee:69:cd:07:9f:d0 33:33:00:00:00:02 2
packet in 1 ee:69:cd:07:9f:d0 33:33:00:00:00:16 2
```

Figure 4.8 (a): Run the Master Controller in a Terminal at Port Number 6653



```
dell@dell-Inspiron-3576:~/Desktop/SDN$ ryu-manager --ofp-tcp-listen-port 6654 l3switch_slave.py
loading app l3switch_slave.py
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app l3switch_slave.py of SimpleSwitch13
This Controller is Slave
This Controller is Slave
This Controller is Slave
█
```

Figure 4.8 (b): Run the Slave Controller in a Terminal at Port Number 6654

```
dell@dell-Inspiron-3576:~/Desktop/SDN$ sudo python newtopo.py
*** Creating network
*** Adding hosts:
h1 h2
*** Adding switches:
s0 s1 s2
*** Adding links:
(h1, s1) (h2, s2) (s0, s1) (s0, s2)
*** Configuring hosts
h1 h2
*** Starting controller
c0 c1
*** Starting 3 switches
s0 s1 s2 ...
*** Starting CLI:
mininet> █
```

Figure 4.8 (c): Run a topology by using Python API in the Mininet

```
*** Adding links:
(h1, s1) (h2, s2) (s0, s1) (s0, s2)
*** Configuring hosts
h1 h2
*** Starting controller
c0 c1
*** Starting 3 switches
s0 s1 s2 ...
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=20.4 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.23 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.096 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.092 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.102 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.094 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.126 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.079 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.087 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.095 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.097 ms
^C
--- 10.0.0.2 ping statistics ---
18 packets transmitted, 18 received, 0% packet loss, time 17369ms
rtt min/avg/max/mdev = 0.079/1.289/20.478/4.661 ms
mininet> █
```

Figure 4.8 (d): No Duplicate Packets are generated in Master-Slave Simulation

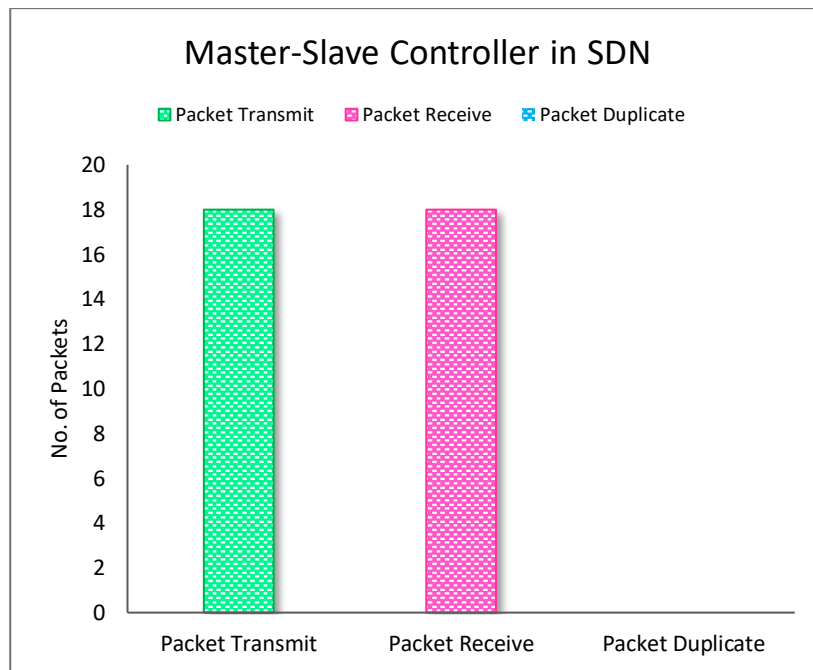


Figure 4.8 (e): Simulation in Master-Slave Controller in SDN

The results of the simulation on the different controller roles in Software Defined Networks are summarised in Table 7.

Table 7: Compare the Result of Controllers in SDN

<b><i>Role of Controller's in SDN Environment</i></b>			
Comparison Parameter	Single Controller	Multiple Controller's	
		Equal Controller	Master-Slave Controller
Risk of central point of failure	Yes	No	No

Network failure	Yes	No	No
Percentage of packet loss	Yes	No	No
Induce duplicate packet	No	Yes	No
Requirement of database synchronous	No	No	Yes
Increase latency of network	Yes	No	No
Scalability of network	No	Yes	Yes

## 4.8 Multi-deployment of Controllers in SDN

Then work on the multi-deployment of controllers in SDN and how their performance is evaluated. For this purpose, iostat and top command are used to evaluate the parameters that affect the performance of the network while using multiple controllers in the network. Figure 4.9 (a and b) show the overall CPU utilization curve fluctuates more in the equal controller as compared to the master-slave controller.

Similarly, Figures 4.10 (a and b) show overall memory utilization in both multiple controllers and Figure 4.11 shows the comparison of both roles of multiple controllers in CPU utilization respective. Thus, the master-slave controller is preferable to an equal controller for the proper utilization of resources.



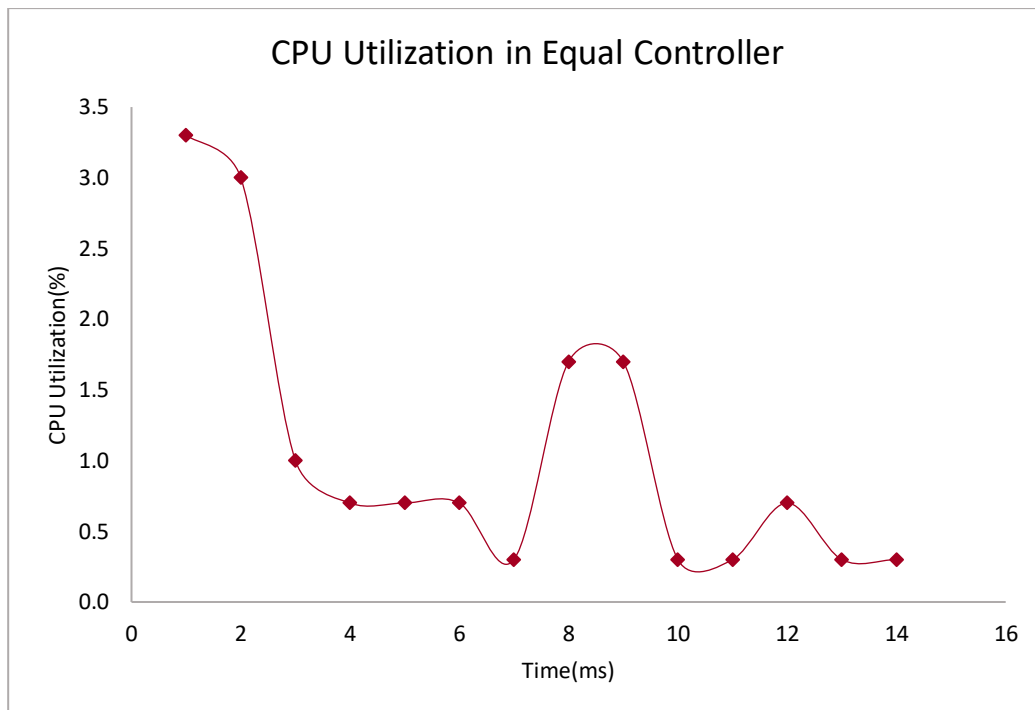


Figure 4.9 (a): Overall CPU Utilization in Equal Controller

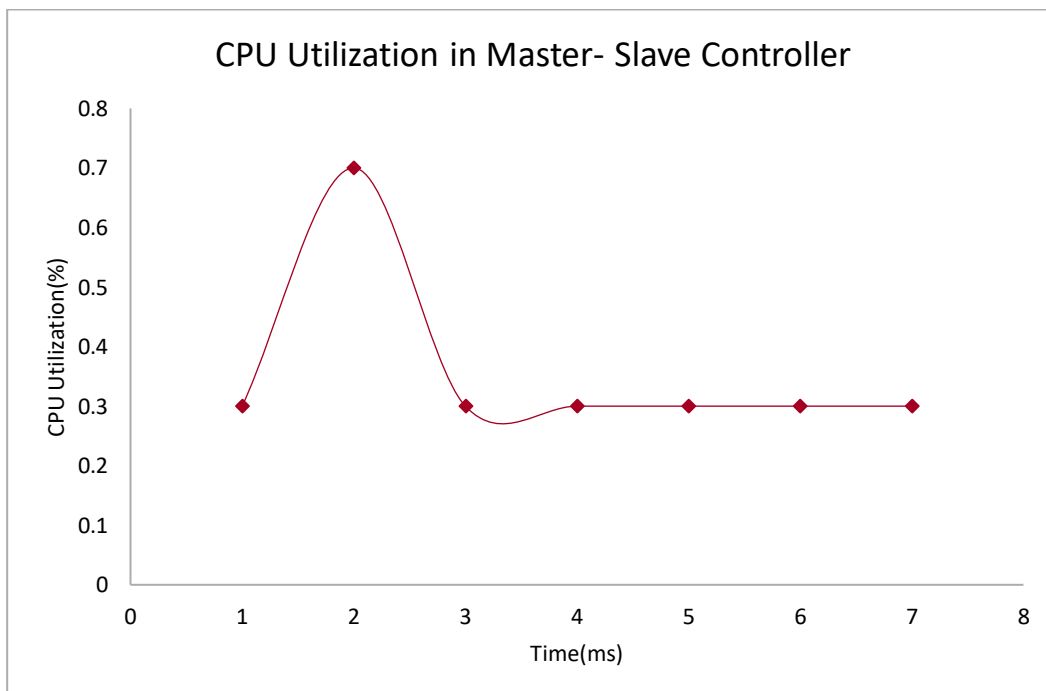


Figure 4.9 (b): Overall CPU Utilization in Master-Slave Controller

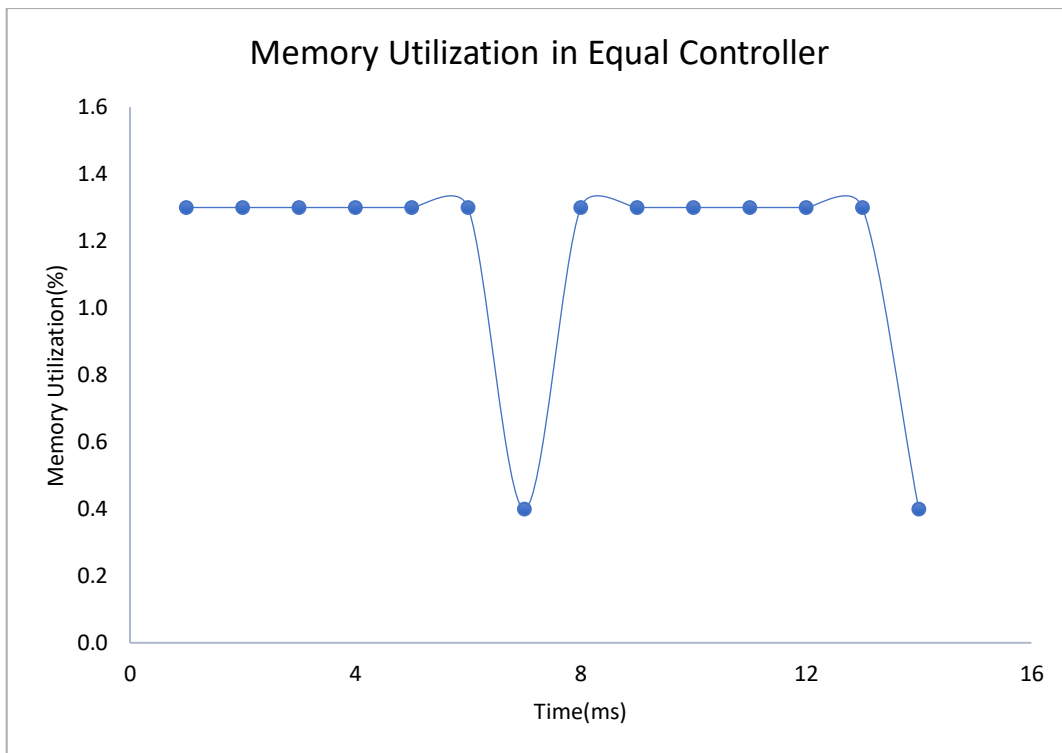


Figure 4.10 (a): Overall Memory Utilization in Equal Controller

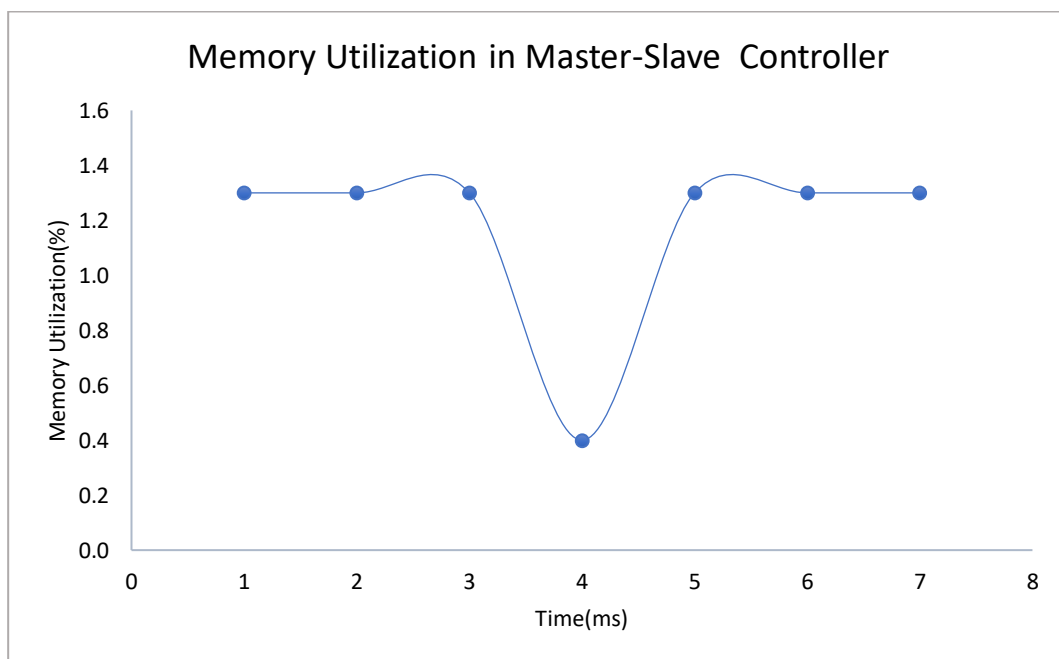


Figure 4.10 (b): Overall Memory Utilization in Master-Slave Controller

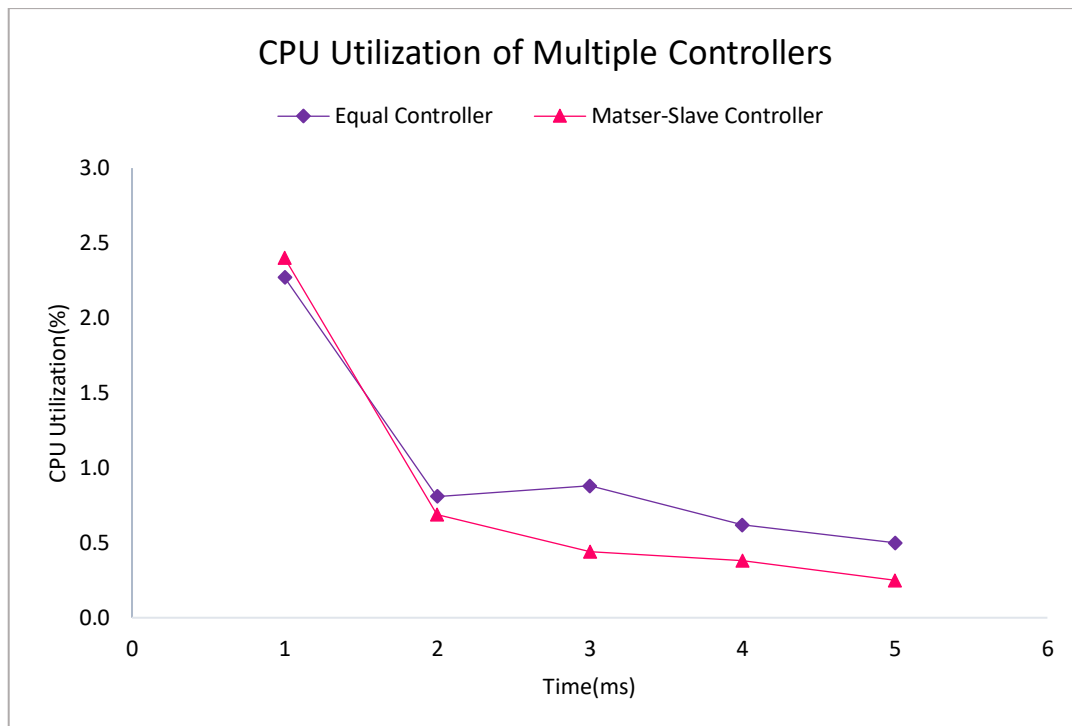


Figure 4.11: Comparison of CPU Utilization in Multiple Controller

Now to compare the various role of SDN controllers' performance in terms of the round-trip time (RTT). During simulation to evaluate the result of SDN controllers is summarized in Table 8 and shown in Figures (4.12 (a) to 4.12 (d)).

Table 8: Performance Metrics of Controllers w.r.t. Round-trip Time

<b><i>Round-Trip Time (RTT)</i></b>	<b><i>Min (ms)</i></b>	<b><i>Avg (ms)</i></b>	<b><i>Max (ms)</i></b>	<b><i>Mdev (ms)</i></b>
<b><i>Single Controller</i></b>	0.042	43.522	2062.932	268.729
<b><i>Equal Controller</i></b>	0.081	41.889	94.402	32.708
<b><i>Master-Slave Controller</i></b>	0.073	0.473	36.974	3.670

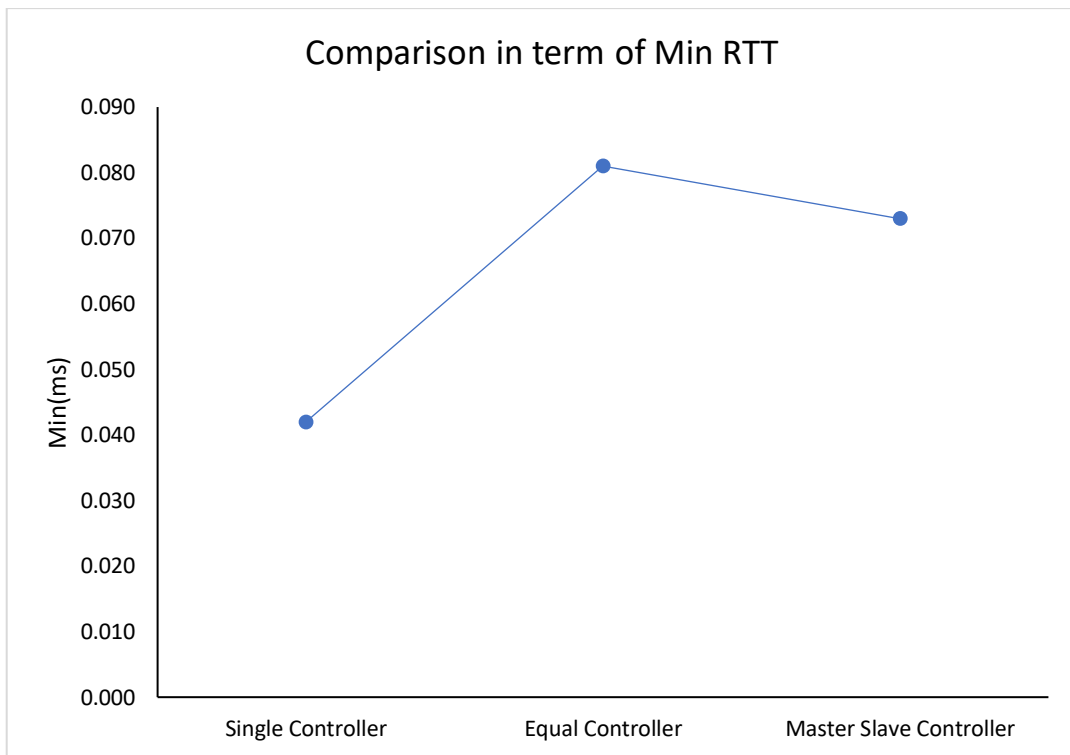


Figure 4.12 (a): Performance Metrics of Controllers in term Minimum RTT

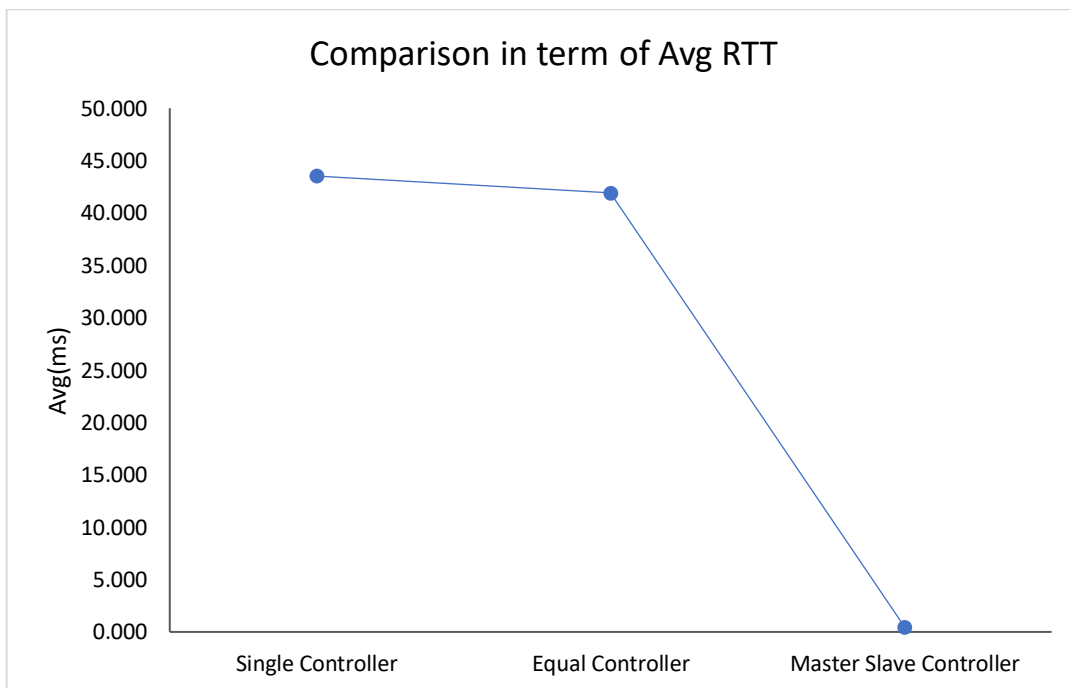


Figure 4.12 (b): Performance Metrics of Controllers in term Average RTT

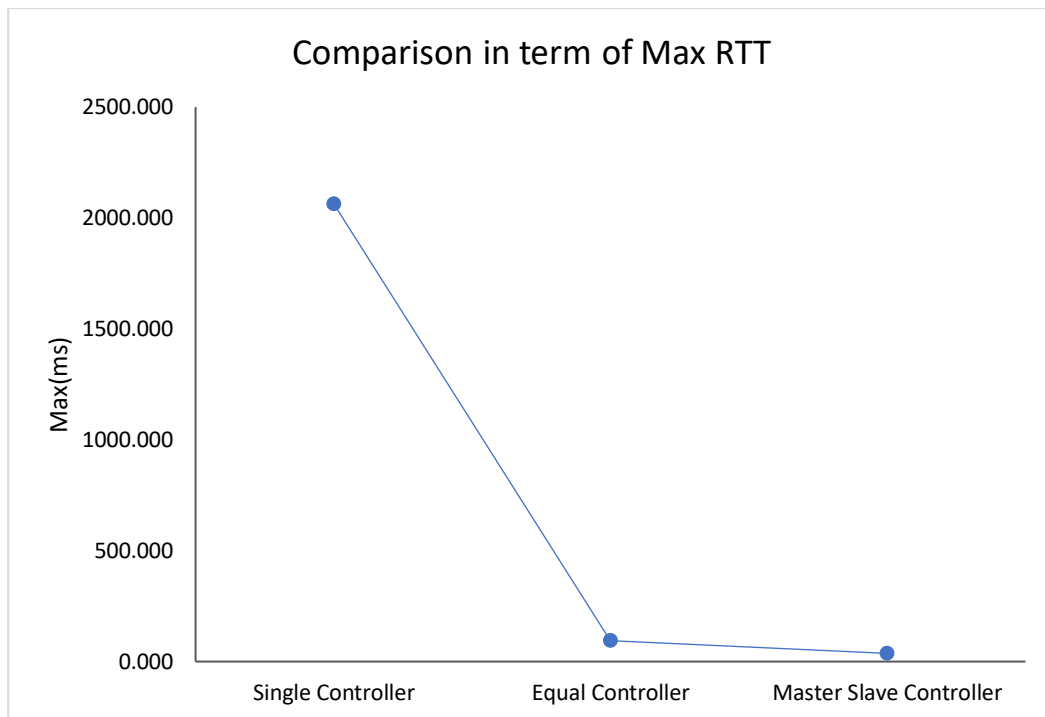


Figure 4.12 (c): Performance Metrics of Controllers in term Maximum RTT

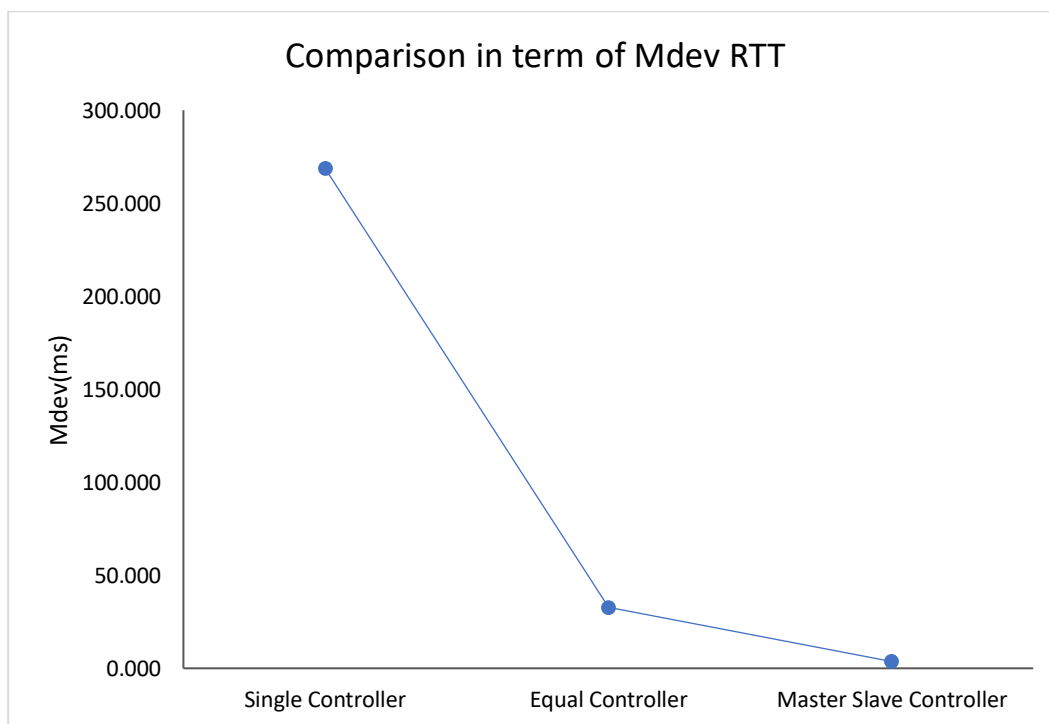


Figure 4.12 (d): Performance Metrics of Controllers in term Mean Deviation RTT

To evaluate the effectiveness evaluation of SDN controllers on the role using the iperf tool. As shown in Figures 4.13 (a) and (b), respectively, Table 9 compares controllers based on various metrics or parameters, including average throughput, average bandwidth, and ping delay. This demonstrates that a master slave controller performs better than other kinds of controllers.

Table 9: Comparison of Controllers w.r.t. Average Throughput, Average Bandwidth and Ping Delay

<i>SDN Controller's</i>	<i>Average Throughput (GBytes)</i>	<i>Average Bandwidth (Gbits/sec)</i>	<i>Ping Delay (ms)</i>	<i>Time Interval (Sec)</i>
<i>Single Controller</i>	63.5	36.4	43.480	15
<i>Equal Controller</i>	69.8	40.0	41.808	15
<i>Master Slave Controller</i>	70.9	40.6	0.400	15

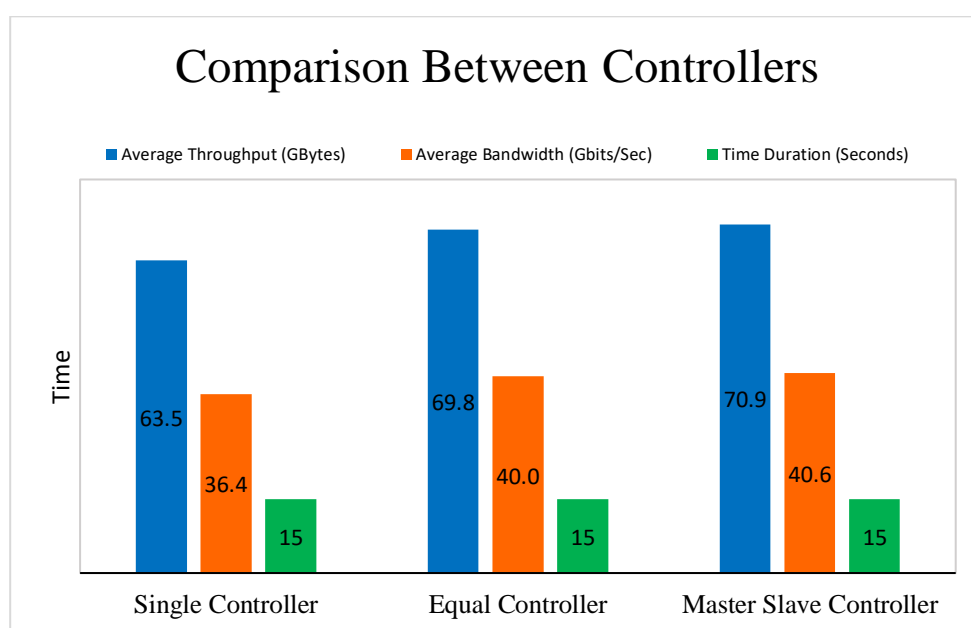


Figure 4.13 (a): Comparison between Controllers w.r.t. Average Throughput, Average Bandwidth and Time Duration

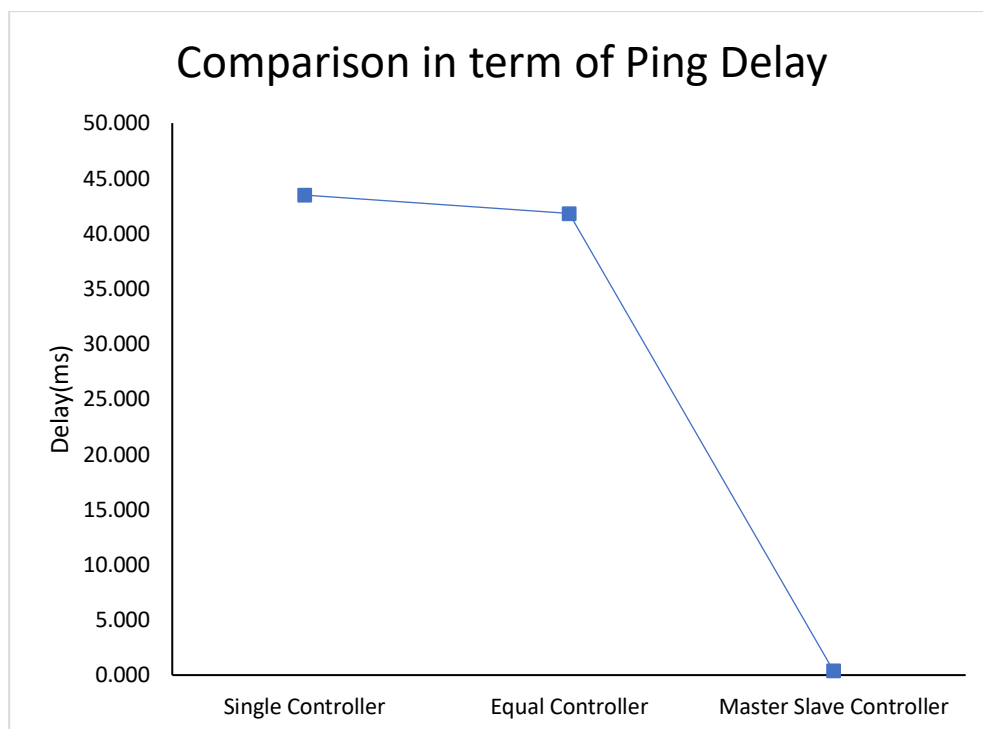


Figure 4.13 (b): Performance of Controllers w.r.t. Ping Delay

## 4.9 Conclusion

Software Defined Network decouples the control plane from the data plane. Due to this separation, a controller provides a centralized logical view of a whole network. As a result, it raises the network's utmost risk of a single point of failure. To eliminate the risk of SPOF in SDN by using multiple controllers. In the simulation, to analyze when a single controller fails, then the percentage of packet loss is increasing. To overcome this problem, by using multiple controllers in SDN. No packet loss occurs in the equal controllers but, they generate duplicate packets in a network. To avoid duplicate packets in the network then use the master-slave configuration in SDN. During simulation, Tables 8 and 9 demonstrate that a master slave controller performs better than other kinds of controllers. But a data synchronous problem has occurred between the master and slave controllers. In the future, try to maintain data synchronization between the master and slave controllers to enhance the availability of the networks.

# **Chapter 5**

**Load Balancing using  
Queuing Models in  
Multiple Controller  
Environment of SDN**



# **5 CHAPTER**

## **Load Balancing using Queuing Models in Multiple Controller Environment of SDN**

### **5.1 Introduction**

A significant part of a software defined network is the supervisor. Control and data planes were separated by the SDN model. A controller now serves as the central hub for all management operations. Now all control logic is supervised by a controller. In SDN, the main responsibilities of a controller are policy enforcement, network configuration, inserting or modifying flow rules in a table, maintaining and updating topology information (topology management), taking the decision regarding how to control network traffic in a network, and so on. Thus, a controller provides global information about a whole network. Moreover, it becomes convenient for operators and researchers to adjust and extend the network flexibility to the added new feature of network function through programmability. As a consequence, a higher impact or risk factor of a failure is increasing on a single controller in SDN. When a single controller fails, then the entire network becomes collapse. By using numerous controllers to mitigate the risk of a single point of failure in SDN. The maximum effort has been dedicated to the controller for achieving high performance, scalability, reliability, and availability of services.

## **5.2 Related Work**

In distributed environments, controllers have addressed various challenges concerning taking into account problems like consistency, balancing loads, tolerance for faults, and latency time. But centralized controller mainly suffers issues are scalability, reliability, and so on. A centralized controller poses a threat like a single point of failure. It is not able to recover automatically from this failure. So, the main concern to achieving maximum resilience and scalability of multiple controllers in a cost-effective way [6,64,66].

When a controller installs a lot of new flow rules in switches, there is a substantial overhead at both the control and data planes in SDN [24,66]. To avoid a bottleneck in SDN must consider a tradeoff between latency and load balancing. So, need to manage load balancing in SDN because a controller cannot work efficiently in a large-scale network as well as increase the traffic flow. Therefore, loss of packets is the increase in a network and degrades the performance of a network. Moreover, the maximum chance of failure occurs in SDN. Multiple controllers suggested to maintaining the tradeoff between these metrics' latency, reliability, and load balancing [24,64]. But a critical problem arises in the case of multiple controllers to determine an optimal number of controllers and their locations [24, 66]. In a distributed environment, multiple controllers are used in a network then some issues are needed to get more attention. Resolve these issues as soon as possible otherwise degrades the network performance exponentially in a network. Such issues are network latency and congestion, imbalance in the load between the multiple controllers. So, the load balancing between multiple controllers is a crucial matter; for optimal utilization of resources in the network that decreases the overheads in the control plane [49,64-95]. It is accomplished by distributing a load of the overloaded controller to other controllers of the network. Unbalancing between the controllers reduces the overall utilization of resources in a network [92,93]. As a consequence, some controllers in the network reach their performance bottleneck due to the increase in delay of response whereas some other controllers are underloaded in a network.

## **5.3 Problem Formulation**

Load balancing is a vital issue in the case of multiple controllers in order to guarantee optimal network resource usage and prevent reducing the overheads in the control plane [92]. To achieve the load balancing between controllers, need to distribute the load of the overloaded controller to other controllers of the network. The imbalance between controllers reduces the overall resource utilization of a network [92-94]; because some controllers reach their performance bottleneck which increases the response delay, whereas other controllers are underloaded or idle state in a network [88,89].

In SDN, the controller is a very crucial component because it manages all traffic of the network. While forwarding devices serve as a basic forwarding element in the data plane, controller is responsible for making all of the network's decisions regarding routing. When a controller handles more network traffic than its capacity [24-76], then several problems occur such as controller failure, controller overload. When it exceeds its threshold value, and cascaded failure of controllers in the network [88]. Whenever multiple controllers are used in Software Defined Network then counter new challenges as the load balancing between multiple controllers, due to the uneven distribution of the network traffic, state synchronization occurs between controllers, resulting in a cascading failure of the controller in the network that increases the latency, and the drop rate of the packet in the network. Moreover, it decreases the performance of the network [86-95].

### **5.3.1 Need of Load Balancing in SDN Controllers**

For optimal network resource utilisation and minimal control layer overhead, load balancing is a critical problem in multiple controllers. To distribute the burden of an overloaded controller to other network controllers in order to accomplish load balancing between multiple controllers. Uneven load distribution among controllers lowers the network's overall resource utilisation [24,85-90]. As a consequence, some controllers are underloaded or idle in the network whereas some controllers reach

their performance to bottleneck and upsurge response delay due to network traffic [71-85].

The actual controller is a crucial component in SDN because it manages all traffic of the network. Because the controller has the authority for making all routing decisions of the network; So, since it directly impacts network performance, balancing the workload across controllers is one of the trickiest tasks. Additionally, the performance of the network is considerably affected by the control plane's scalability and load balancing between them.

## **5.4 Proposed Design of Algorithm for Load Balancing**

The motive of this objective is how to maintain and control the load balancing between multiple SDN controllers. Both fault tolerance and load balancing are complicated and interrelated issues while dealing with the multiple controllers in SDN. To resolve these issues, using the Queuing theory technique and Markov continuous chain helps to handle the fluctuation of load between the multiple controllers. For this purpose, to design a load balancing algorithm by the integrated concept Queuing Theory Technique and Markov Continuous Chain to manage the load balance between the SDN controllers, which reduces packet dropping or packet lost ratio of the network. It is happened in a network due to the lack of load balancing techniques. So, it is necessary to distribute the proper workload on controllers because network traffic fluctuates dynamically.

Network traffic behaves like a stochastic process whose behavior is random changes over time. The Markov process is a simple stochastic process in which the distribution of the future state depends only on the present state of a process rather than how to arrive at the present state. Thus, the stochastic process must be holding the Markov property “memoryless” [75,85,100]. According to this property, the probability of the future state ( $Y_{t+1}$ ) at time instant ( $t+1$ ) depends upon the present state ( $Y_t$ ) at time instant ( $t$ ) has been described as:

$$P [ Y_{t+1} | Y_t, Y_{t-1}, \dots, Y_2, Y_1, Y_0 ] = P [ Y_{t+1} | Y_t ]$$

Where,

$Y_{t+1}$  dependent on  $Y_t$ , but not dependent on  $Y_{t-1}, Y_{t-2}, Y_{t-3}, \dots, Y_2, Y_1, Y_0$ .

The Mathematical Notation of Markov Property as follow:

$$\mathbb{P}(Y_{t+1} = A \mid Y_t = A_t, Y_{t-1} = A_{t-1}, \dots, Y_0 = A_0) = \mathbb{P}(Y_{t+1} = A \mid Y_t = A_t)$$

for all  $t = 1, 2, 3, \dots$  and for states  $A_0, A_1, \dots, A_t, A$

This memoryless property is also applicable in queuing model. It is also known as the network queuing model. In queuing model, two main events have occurred; either the arrival rate or service rate of a packet in the network. The poisson distribution and the exponential distribution are followed, respectively, by the arrival rate, the time between arrivals, and the service rate [18, 19]. The arrival rate of a packet in a system at time instant  $t$  is represented as  $\lambda$  and the service rate of a packet in a system (or processing of packet) at time instant  $t$  is represented as  $\mu$ . The traffic intensity or utilization factor of a controller is represented as  $\rho$ . To calculate the value of traffic intensity is  $\rho = \lambda / \mu$ ; idle time of a system is represented by  $P_0 = 1 - \rho$ . Figure 5.1 shows how queuing technique is applicable in the SDN controller. Figure 5.2 depicts the general layout of an algorithm, while Figure 5.3 depicts the flowchart.

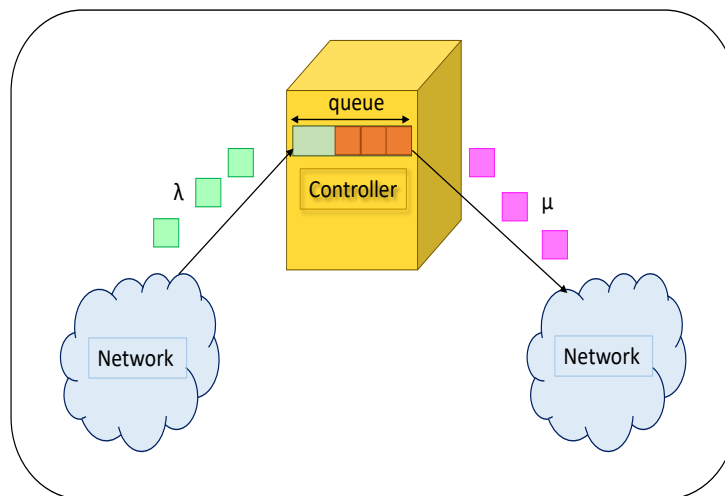


Figure 5.1: Use of Queuing concept in SDN Controller

**Algorithm for Load Balancing in SDN for Multiple Controllers by using Queuing Technique**

Initial Requirement:

CC is represented for Current Controller; SC is represented for Slave Controllers in the network;

$\lambda$  = Arrival Rate of the packet;  $\mu$  = Service Rate of the packet;  $f$  = Traffic Intensity of Controller;

Traffic Intensity ( $f$ ) acts as a threshold value of a Controller in the network.

Result:

0: No need for load balancing

1: Successfully load balancing perform

***/\* Load Balancing between Multiple Controllers in SDN\*/***

if Load of CC >  $f$  then

{

    for (i=0; i<n; i++)

    {

*/\* Calculate the traffic intensity value of all slave controllers SC<sub>i</sub>\*/*

$$f_i = \lambda_i / \mu_i$$

        Select controller SC<sub>i</sub> which has the lowest traffic intensity value in the network.

    }

    return 1;

}

else

{

    return 0;

}

Figure 5.2: Algorithm for Load Balancing in Multiple Controllers

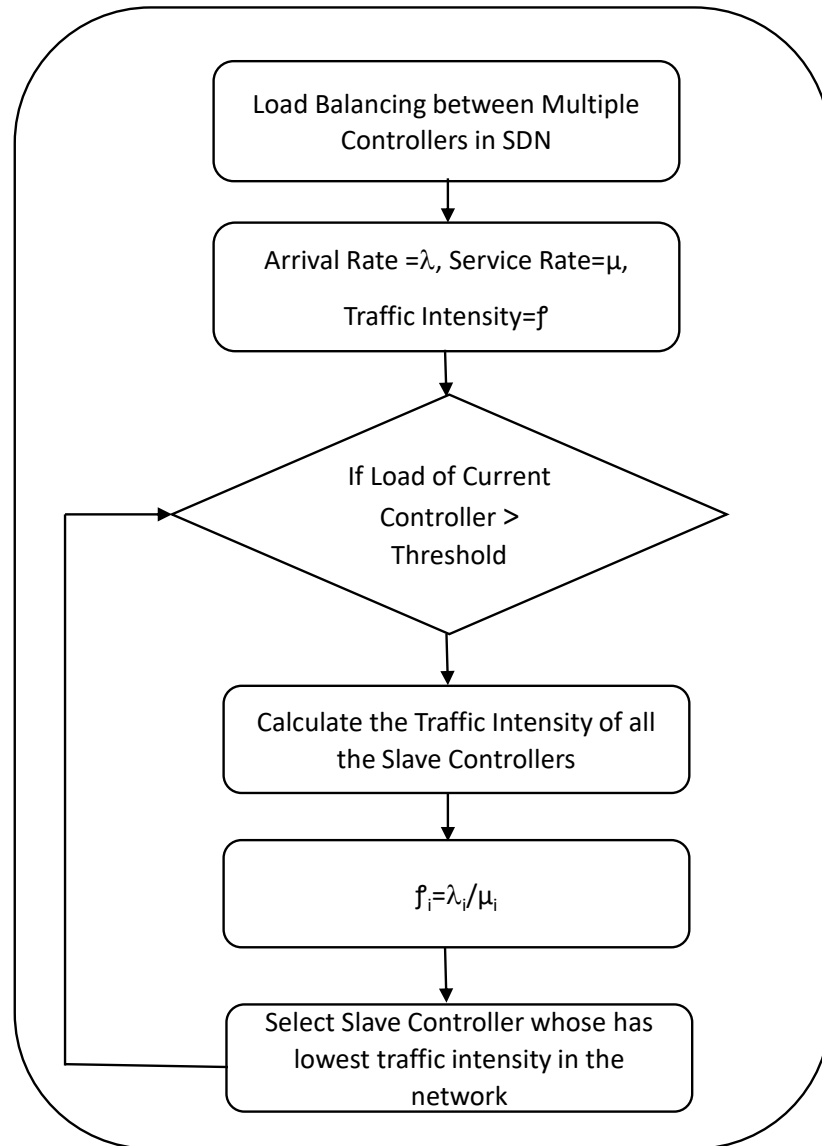


Figure 5.3: Flowchart for Load Balancing in SDN Controllers

## 5.5 Queuing Model M/M/1: $\infty/\infty$ versus M/M/1: N/ $\infty$ and its Simulation

In queuing model, it is expressed as the sum of three independent compound probabilities [75,85,100] as depicted in Figure 5.4 (a and b) respectively. After performing all the required manipulations in the M/M/1 queue model with infinite

and finite queue length, which also highlights the impact on different parameters, including queue length, system length, waiting time in the queue, and system.

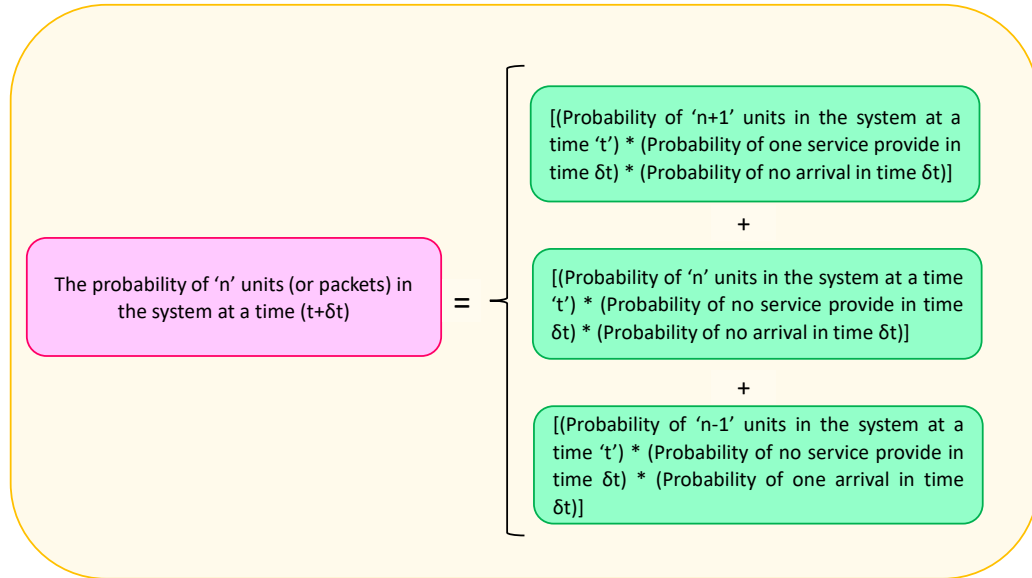


Figure 5.4 (a): Probability of events occur in time interval (t+δt)

To obtain the steady-state of the differential equation of the M/M/1: ∞/∞ model is the product of three possibilities events occur [100] are shown in Figure 5.4 (a), Figure 5.4 (b), and the representation of both queue models show in Figure 5.4 (c) and 5.4 (d). Therefore,

$$\frac{\delta P_n(t+\delta t)}{\delta t} = \{-(\lambda + \mu)P_n(t) + \lambda P_{n-1}(t) + \mu P_{n+1}(t); \text{ for } n > 0\} \quad (5.1)$$

$$\frac{\delta P_0(t+\delta t)}{\delta t} = \{-\lambda P_0(t) + \mu P_1(t); \text{ for } n = 0\} \quad (5.2)$$

To solve the above differential equations of the queuing model M/M/1: ∞/∞, to find the value of P<sub>1</sub> from equation (5.2).

$$P_1 = \lambda/\mu$$

Then put n=1 in equation (1) and get P<sub>2</sub> = (λ/μ)<sup>2</sup>P<sub>0</sub> and so on.



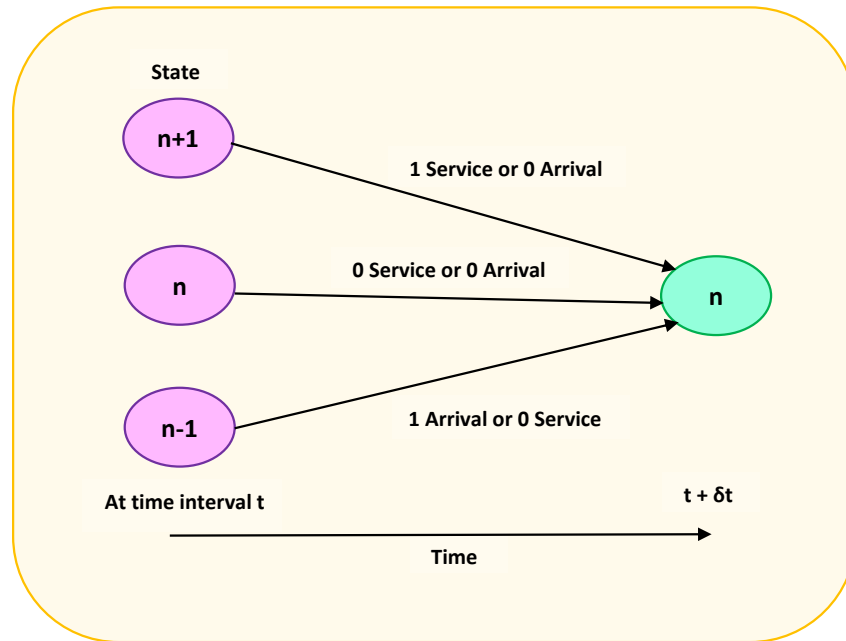


Figure 5.4 (b): Possible of events occur in time interval (t+δt)

Similarly,

$$P_n = (\lambda/\mu)^n P_0$$

Or

$$P_n = f^n P_0 \quad (5.3)$$

After calculating the probability of  $P_n$  of 'n' packets in the system and the probability that the queuing system is idle by  $P_0$ . The queuing system also provides four important properties which are related to each other [100]:

- Calculate the length of the system ( $L_s$ ) by using equation (5.4).

$$L_s = f/(1 - f) \quad (5.4)$$

- To calculate the length of queue ( $L_q$ ) by using equation (5.5).

$$L_q = L_s * f \quad (5.5)$$

- Using equation 5.6 to determine the average waiting time of a packet in a system ( $W_s$ ).

$$W_s = L_s/\lambda \quad (5.6)$$

- Using equation 5.7, calculate the average waiting time of packets in the queue ( $W_q$ ).

$$W_q = L_q/\lambda \quad (5.7)$$

But in M/M/1:  $\infty/\infty$  queuing model creates infinite queue length which also affects the value parameters like length of queue ( $L_q$ ), length of system ( $L_s$ ), waiting time in queue ( $W_q$ ), and waiting time in system ( $W_s$ ) in Figure 5.5. Thus, it is preferable to use M/M/1: N/ $\infty$  queuing model in which the length of the queue is finite. As a result, the number of arrivals won't go over N in any situation. As a result, the system's capacity is capped or limited at let's say N.

Let

$$\text{Arrival Rate } (\lambda) = \lambda_n$$

$$\lambda_n = \begin{cases} \lambda, & \text{if } n = 0, 1, 2, \dots, N - 1 \\ 0, & \text{if } n \geq N \end{cases}$$

and

$$\text{Service Rate } (\mu) = \mu_n$$

$$\mu_n = \{ \mu, \text{ for } n = 1, 2, 3, \dots \}$$

Similarly, solve differential equations of the queuing model M/M/1: N/ $\infty$ , and get the value of  $P_0$  and  $P_n$  in equations (5.8) and (5.9).

Therefore,

$$P_0 = \left[ \frac{1-f}{1-f^{N+1}} \right] \quad (5.8)$$

$$P_n = f^n P_0 \quad \{ \text{for } n = 0, 1, 2, 3, \dots, N \} \quad (5.9)$$

Similarly, evaluate expression of  $L_s$ ,  $L_q$ ,  $W_s$  and  $W_q$  in queuing model M/M/1: N/ $\infty$  are given below: -

- Calculate the length of the system ( $L_s$ ) by using equation (5.10).

$$L_s = \frac{f}{1-f} \left[ \frac{1+Nf^{N+1}-(N+1)f^N}{1-f^{N+1}} \right] \quad (5.10)$$

- The effective arrival rate ( $\lambda_{eff} = \lambda (1 - P_N)$ )
- To calculate the length of queue ( $L_q$ ) by using equation (5.11).

$$L_q = L_s - \frac{\lambda_{eff}}{\mu} \quad (5.11)$$

- Using equation 5.12 to determine the average waiting time of a packet in a system ( $W_s$ ).

$$W_s = \frac{L_s}{\lambda_{eff}} \quad (5.12)$$

- Using equation 5.13, calculate the average waiting time of packets in the queue ( $W_q$ ).

$$W_q = \frac{L_q}{\lambda_{eff}} \quad (5.13)$$

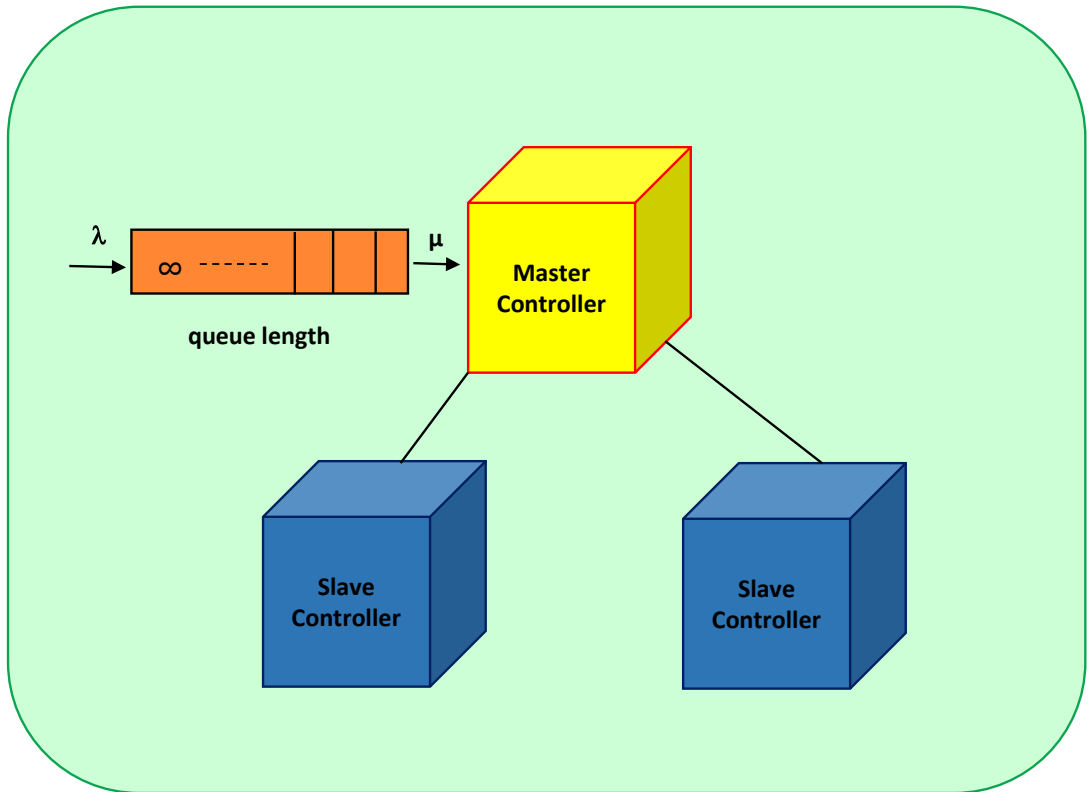


Figure 5.4 (c): Representation of Queue Model M/M/1: ∞/∞ in Network

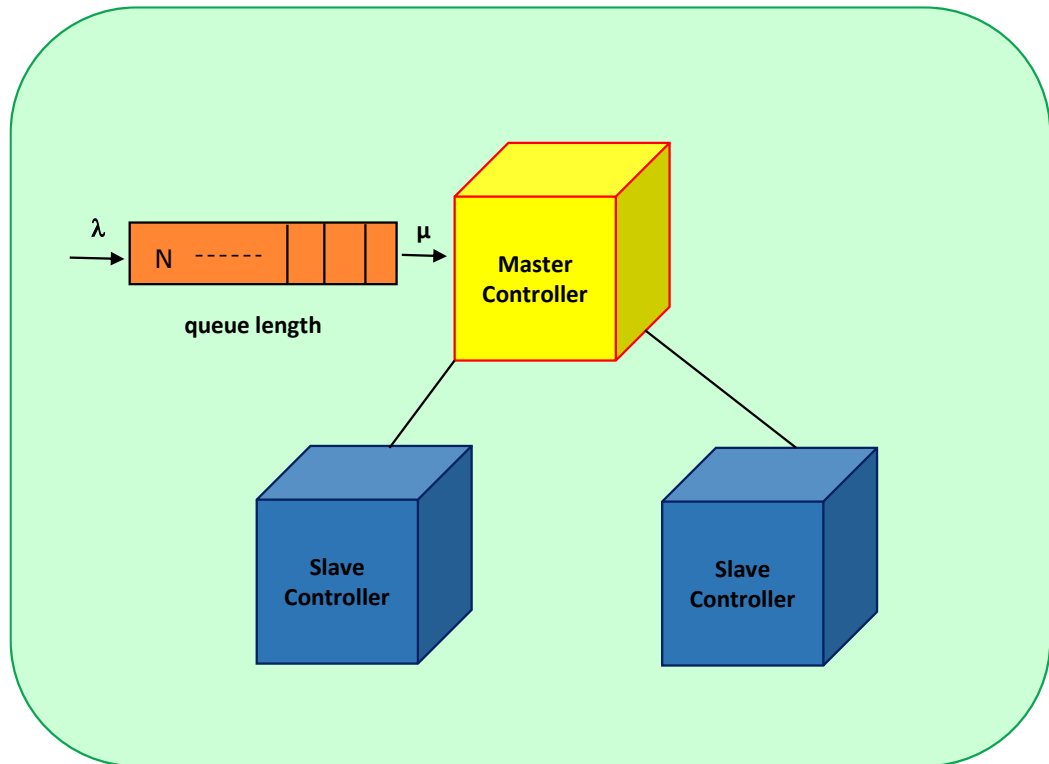


Figure 5.4 (d): Representation of Queue Model M/M/1: N/∞ in Network

In the simulation environment, M/M/1 queue model is simulated with both infinite and finite queues of length, to observe and record the effect of various parameters like length of the system, length of the queue, waiting time in the system, and waiting time in the queue on the overall working of the network. The probability of idle time ( $P_0$ ) is 11% in M/M/1:  $\infty/\infty$  model and 15 % in M/M/1: N/∞ model. After recording observation under a scenario, the results of both queue models (infinite and finite queue length) are compared and plotted in Figure 5.5 and Table 10;

The blue color curve represents M/M/1:  $\infty/\infty$ , and the orange color curve represents M/M/1: N/∞ queue model respectively. The probability of these parameters is  $L_s=8.09\%$ ,  $L_q=7.2\%$ ,  $W_s=1.011\%$  &  $W_q=0.9\%$  approximately in M/M/1:  $\infty/\infty$  model. Similarly, the probability in M/M/1: N/∞ model are  $L_s=3.87\%$ ,  $L_q=3.03\%$ ,  $W_s=0.508\%$  &  $W_q=0.39\%$  approximately. This parameter shows M/M/1: N/∞ model is preferable to another model; because the value of these parameters ( $L_s$ ,  $L_q$ ,  $W_s$  &  $W_q$ ) are less as compared to another model.

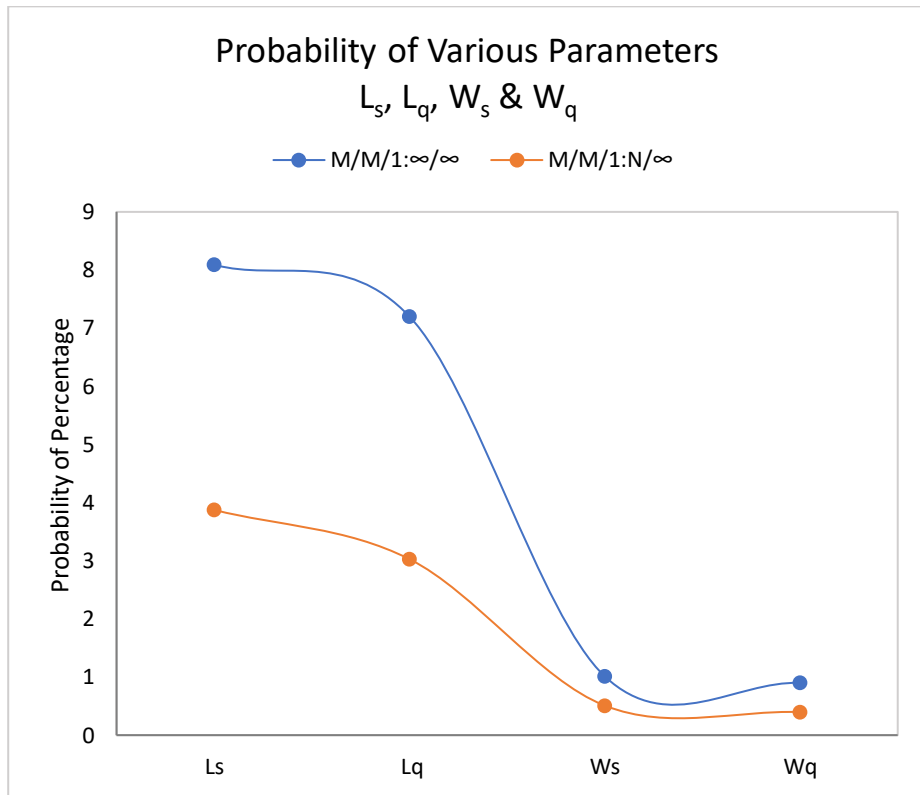


Figure 5.5: Compare various parameters L<sub>s</sub>, L<sub>q</sub>, W<sub>s</sub> & W<sub>q</sub> in both Models

Table 10: Compare various Parameters in both Models

Parameters	M/M/1: ∞/∞	M/M/1: N/∞
L <sub>s</sub>	8.09%	3.874%
L <sub>q</sub>	7.2%	3.027%
W <sub>s</sub>	1.011%	0.508%
W <sub>q</sub>	0.9%	0.397%

The experimental setup required the following software tools are: Ubuntu 18.04 Desktop as an Operating System, NS2 as a simulator, awk programming, Gunplot. By using these tools created various files such as .tcl, .tr, .nam, .awk, and .plot extension.

### **5.5.1 NS2 Simulator**

Network Simulator Version 2 is known by the initials NS2. It is an event-driven, open-source simulator created primarily for studies on computer communication networks. It is capable of simulating both wired and wireless networks. It was created in C++ and Otcl/Tcl and is an object-oriented, discrete event-driven simulator.

To model and examine the behaviour of computer networks, many people use the discrete event network simulator known as NS2. Both C++ and Otcl (Object-oriented Tool Command Language) were used in the creation of this open-source programme. To construct and manage network items and set up network scenarios in NS2, one uses the Otcl (Object-oriented Tool Command Language), an extension of the TCL (Tool Command Language).

One of the tools for TCL-based animation is the NAM (Network Animator). The programme used packet traces from actual networks and network simulations. The topology layout, packet-level animation, and numerous data inspection tools may all be carried out using this particular tool. The nam files are kept with the .nam extension.

During simulation, a network creates a trace file with the .tr file extension to be saved in the detail information of every event that took place during the simulation.

Data manipulation and report generation are both possible with the scripting language awk. Most pattern processing and scanning is done using awk. It performs the corresponding actions after searching one or more files to see whether any lines fit the specified patterns. The initials of the developers Aho, Weinberger, and Kernighan are combined to form the acronym Awk. Gnuplot is a command-line and

GUI tool that can create two-dimensional and three-dimensional graphs. It is a free, interactive, command-driven utility for function and data plotting.

The hardware environment consists of a PC/Laptop with a 64-bit operating system, an Intel(R) Core (TM) i5-8250U CPU @ 1.60GHz 1.80GHz as a CPU, and 4 GB of RAM. This processor is based on the x64 architecture. Table 11 details the simulation parameters in detail.

Table 11: Simulation Parameters

Operating System	Ubuntu 18.04.2 LTS (Long Term Support version of Ubuntu)
System Specification	x64 Intel(R) Core (TM) i5-8250U CPU
Simulator	NS2 (ns-2.35)
Network Animator	nam 1.1.5
Gnuplot	v 5.2
Programming Language	awk
Bandwidth	1.7-2Mb
Date Rate	10-20ms

Through simulator also analyze the various queue-parameters, instantaneous throughput, instantaneous goodput, and instantaneous delay on both queue models as shown in Figure 5.6 (a to m).

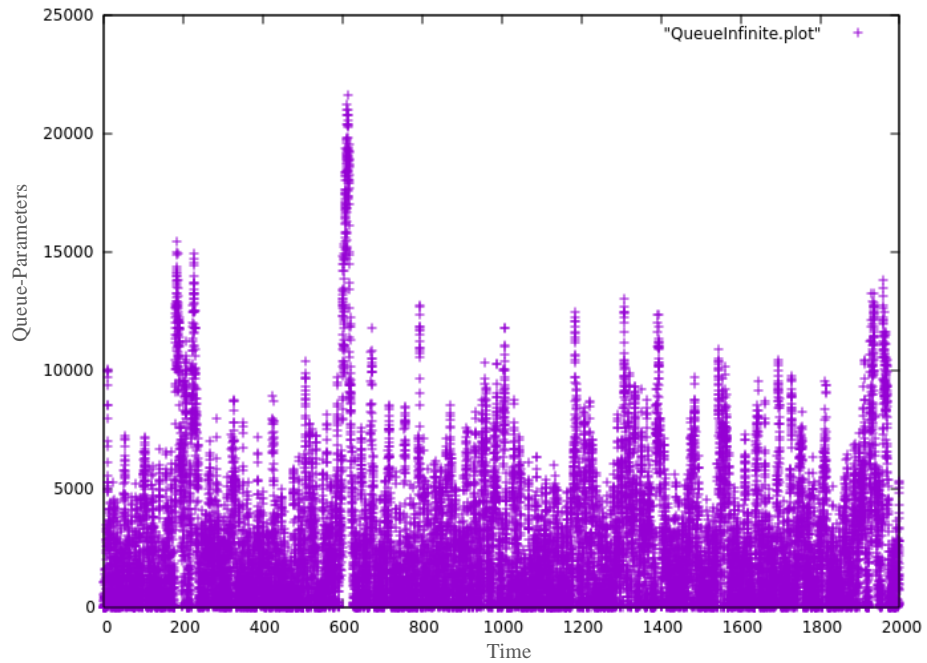


Figure 5.6 (a): M/M/1 Queue Model with Infinite Capacity

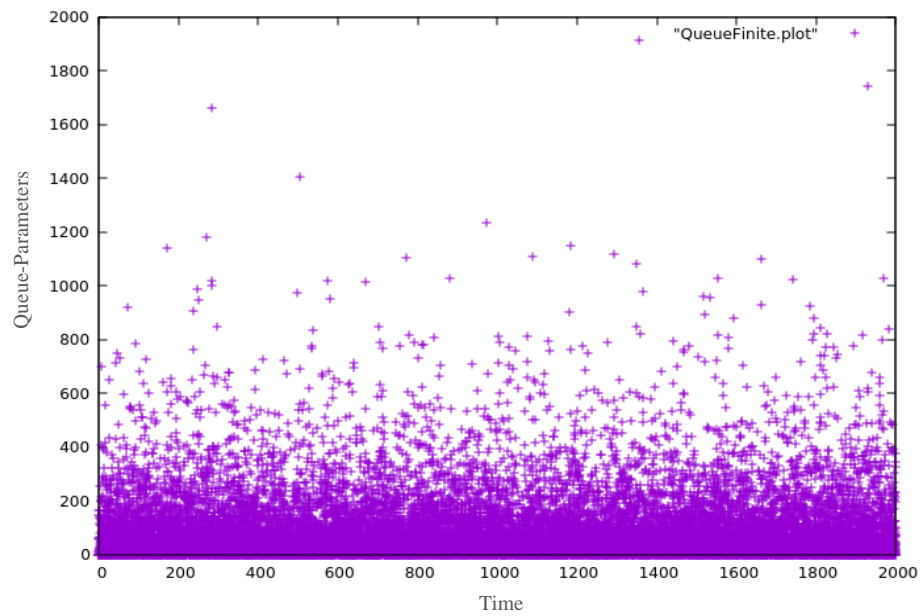


Figure 5.6 (b): M/M/1 Queue Model with Finite Capacity

*\*Note: "Y" axis represents Queue-Parameters like qsizeB, qsizeP, arrivedP, departedP, droppedP, arrivedB, departedB and droppedB; where \*B is number in Bytes, and \*P is number in Packets.*



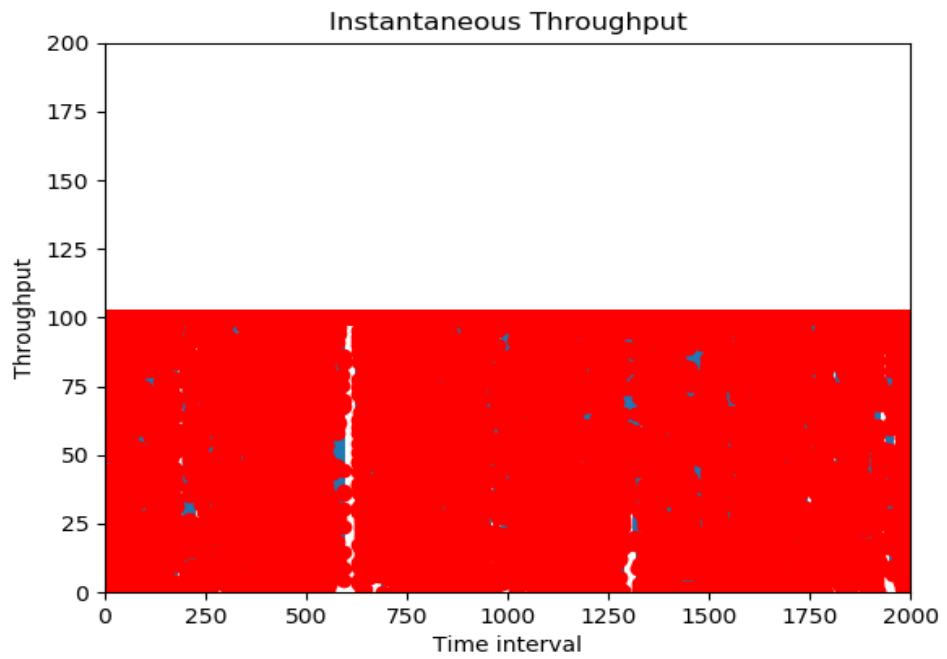


Figure 5.6 (c): Instantaneous Throughput in Queue Model with Infinite Capacity

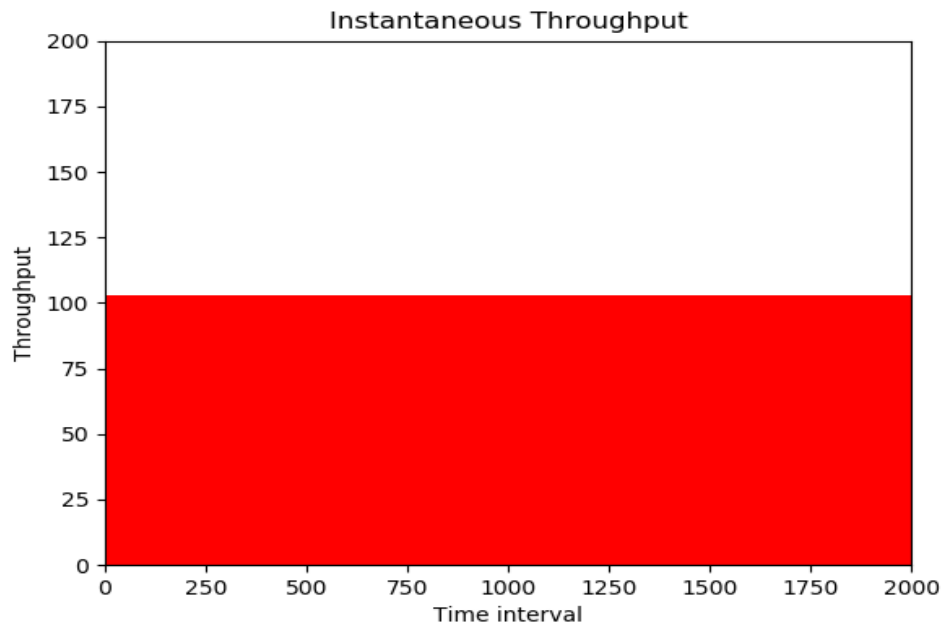


Figure 5.6 (d): Instantaneous Throughput in Queue Model with Finite Capacity

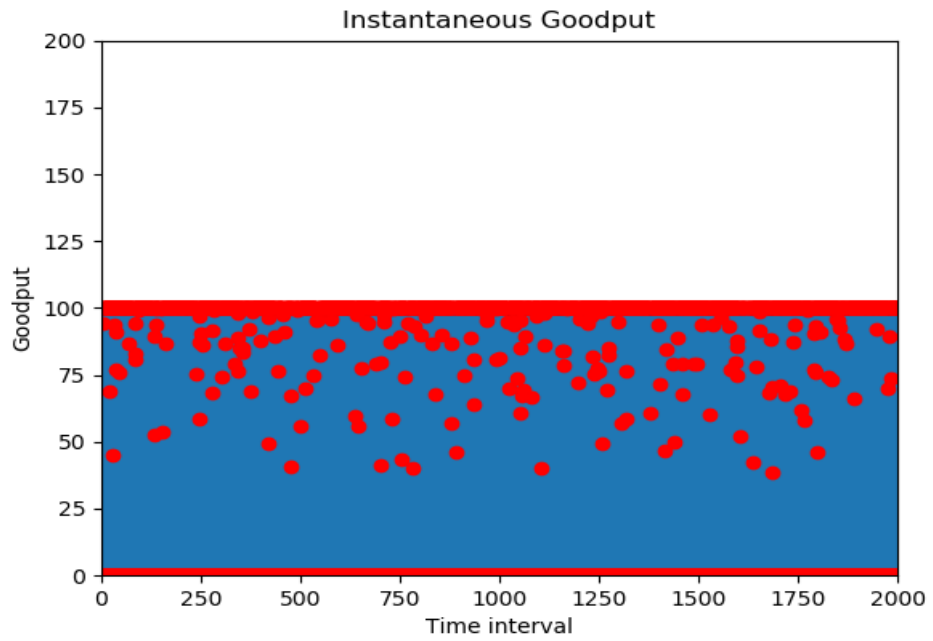


Figure 5.6 (e): Instantaneous Goodput in Queue Model with Infinite Capacity

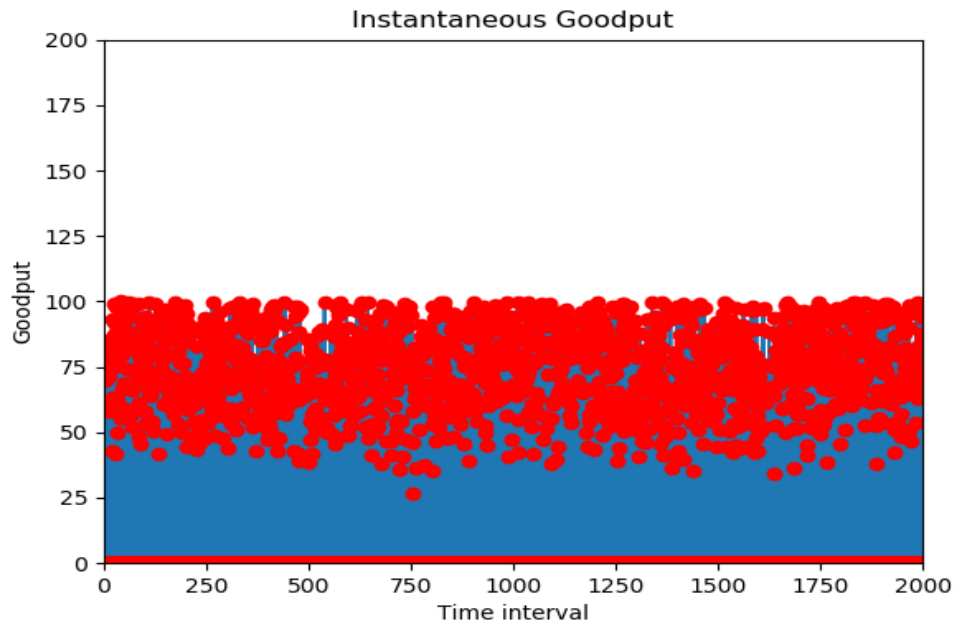


Figure 5.6 (f): Instantaneous Goodput in Queue Model with Finite Capacity

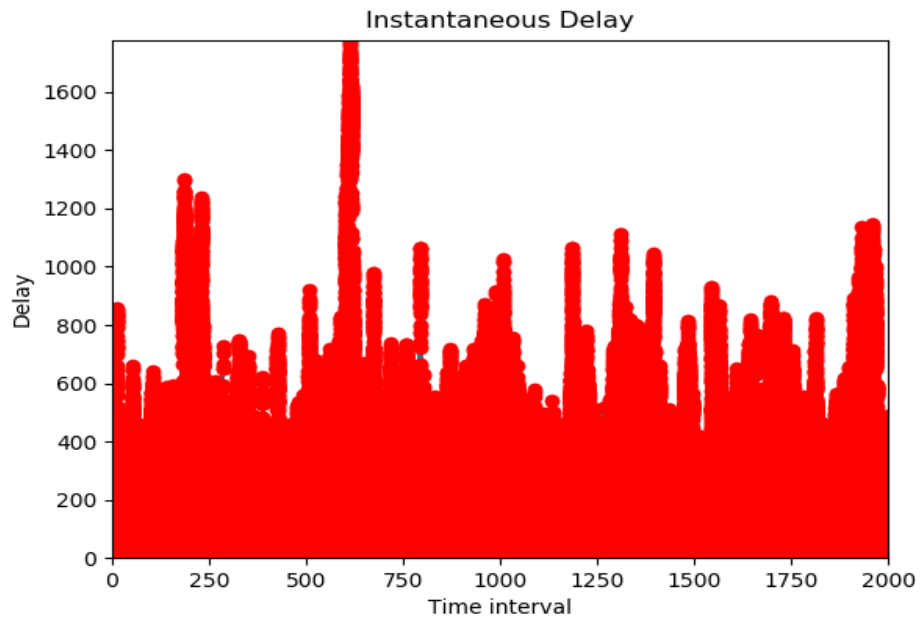


Figure 5.6 (g): Instantaneous Delay in Queue Model with Infinite Capacity

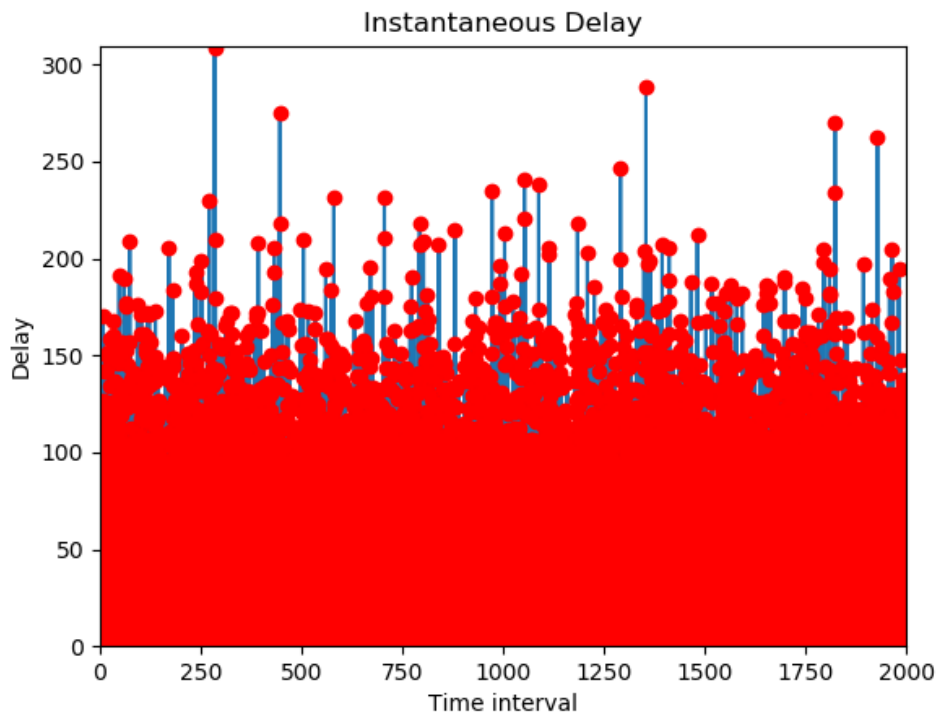


Figure 5.6 (h): Instantaneous Delay in Queue Model with Finite Capacity

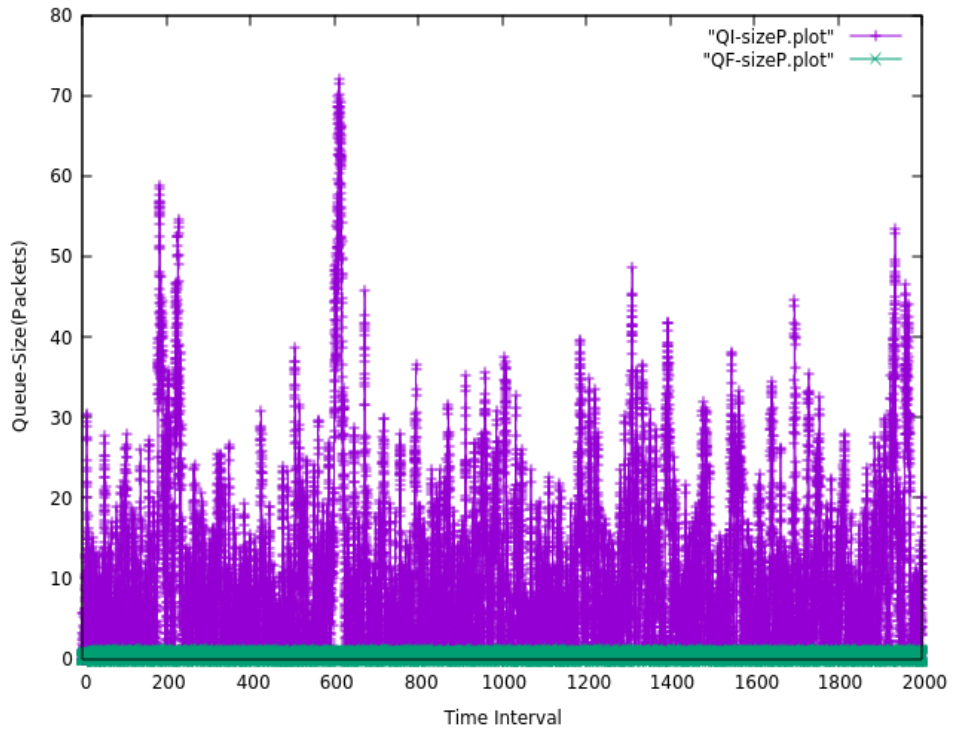


Figure 5.6 (i): Comparison between both Queue Models w.r.t. Queue-Size

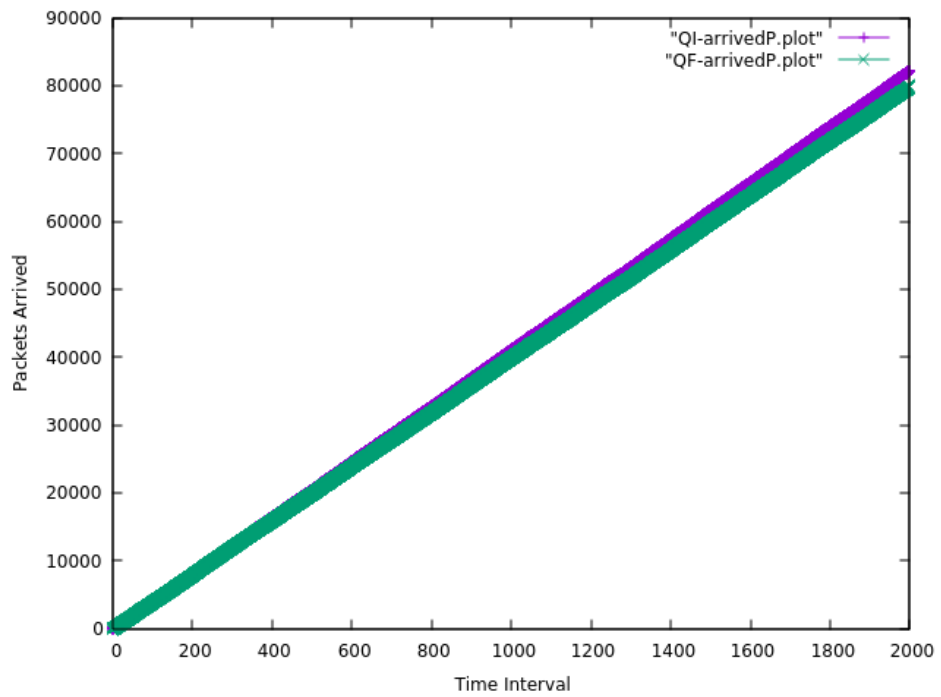


Figure 5.6 (j): Comparison between both Queue Models w.r.t. Arrived Packets

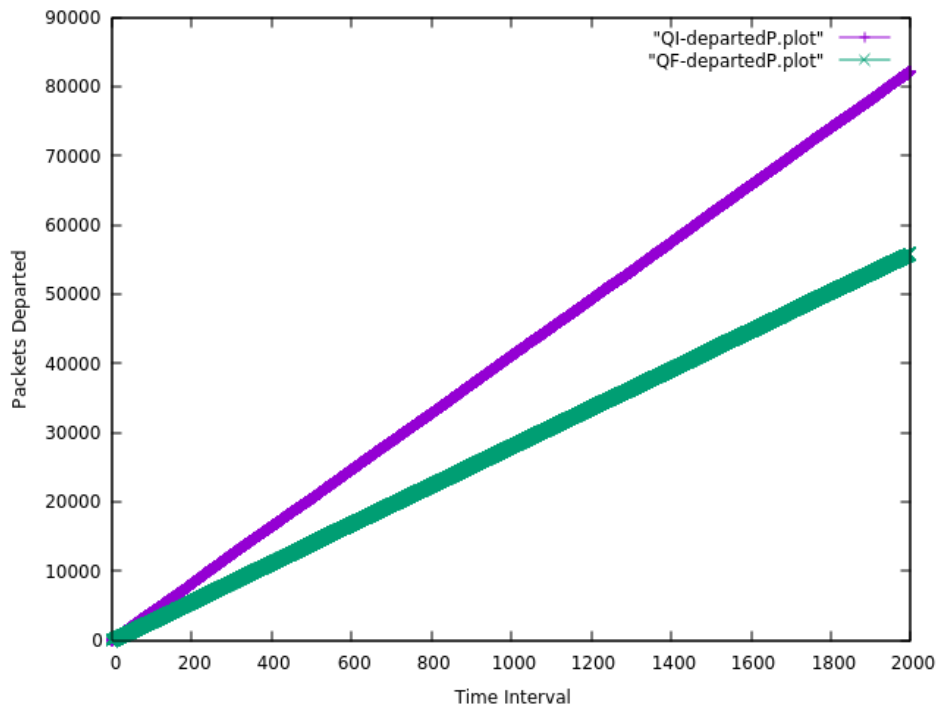


Figure 5.6 (k): Comparison between both Queue Models w.r.t. Departed Packets

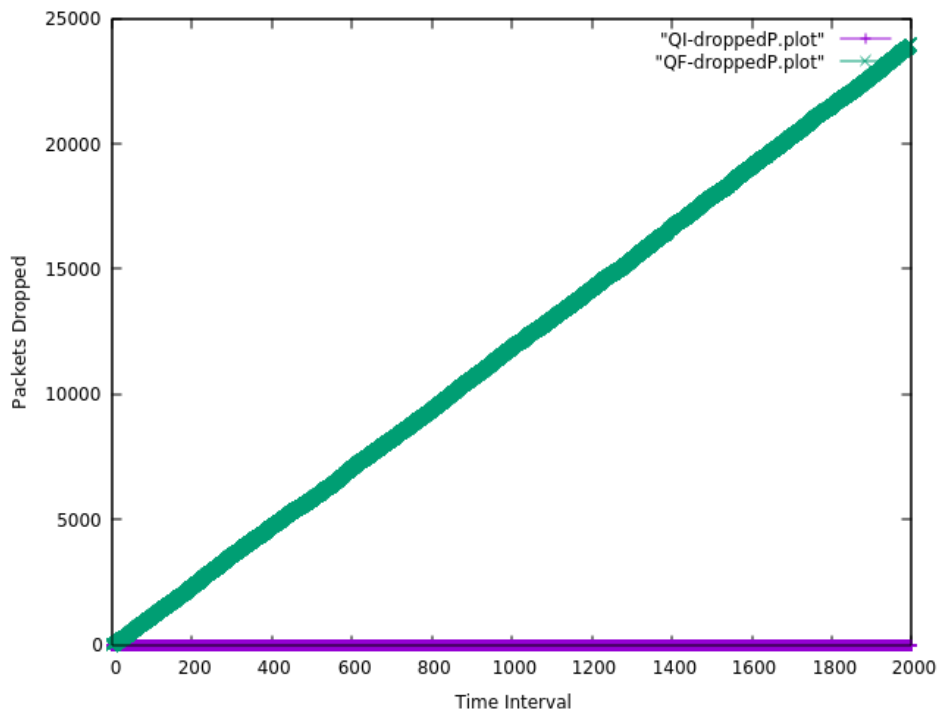


Figure 5.6 (l): Comparison between both Queue Models w.r.t. Dropped Packets

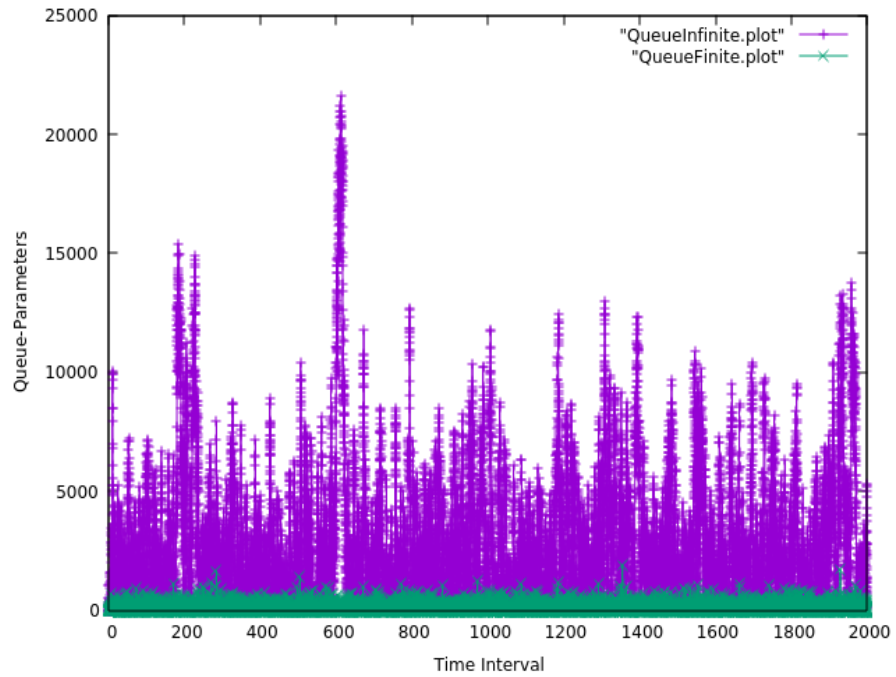


Figure 5.6 (m): Comparison between both Queue Models

\*Note: “Y” axis represents Queue-Parameters like  $qsizeB$ ,  $qsizeP$ ,  $arrivedP$ ,  $departedP$ ,  $droppedP$ ,  $arrivedB$ ,  $departedB$  and  $droppedB$ ; where \*B is number in Bytes, and \*P is number in Packets.

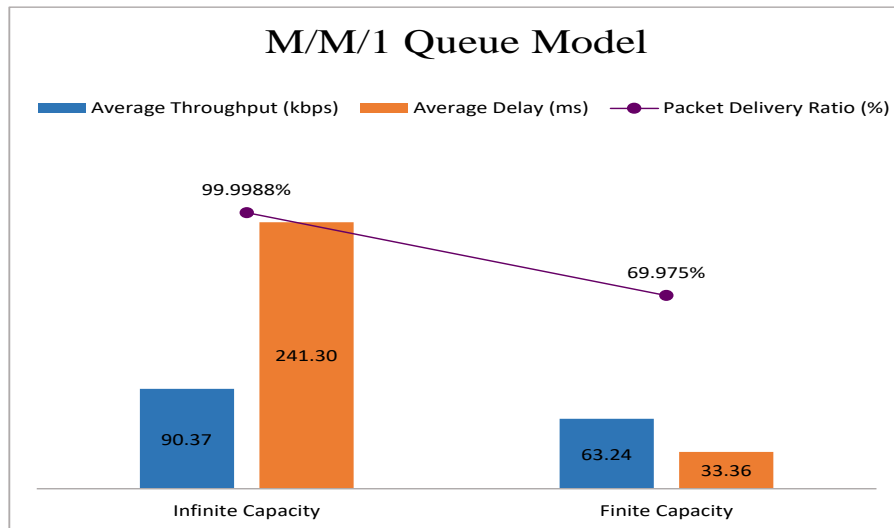


Figure 5.7: Comparison between both Queue Model in Average Throughput, Average Delay and Packet Delivery Ratio

Figure 5.7 shows the average throughput, average delay, and packet delivery ratio in both models; where the blue, orange and purple color highlight respectively. The

average delay of an unlimited queue model is much higher than a limited queue model; because when the size of the queue is increasing or set to infinite then the packet delivery ratio is improved (means increase). But the average delay of the network becomes sequentially increased which is not acceptable for any communication; because it increases the latency of the network that has an adverse effect on network performance. If the size of the queue is set to finite, then the average delay of the network becomes lessened as shown in Figure 5.7.

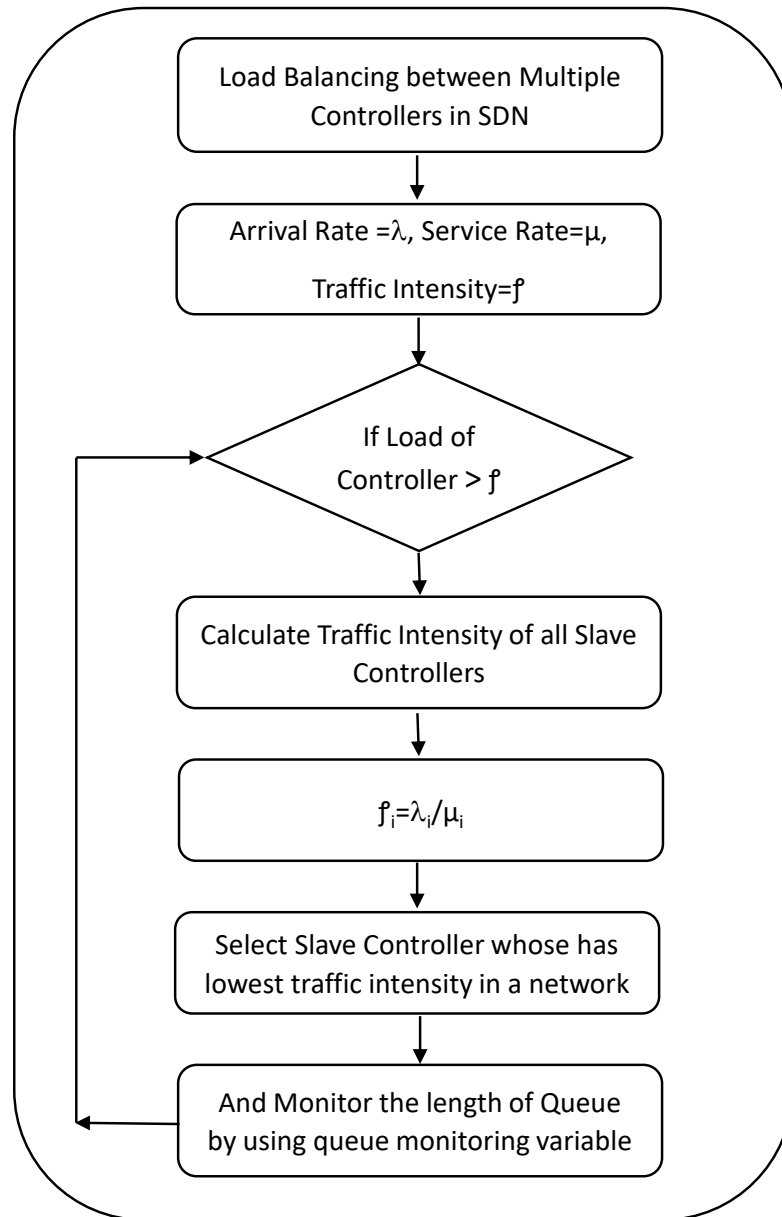


Figure 5.8 (a): Flowchart for Load Balancing in SDN Controllers with queue monitoring variable

As a consequence, the queue model with finite capacity is preferable to the infinity capacity model because the value of these parameters is less in the finite capacity model as compared to another model. Moreover, if a finite length of the queue is selected then the number of arrivals will not exceed then N; because the capacity of a system is limited to say N.

The modified outline of an algorithm and flowchart are shown in Figure 5.8 (a to b) with monitoring queue variable respectively.

**Algorithm for Load Balancing in SDN for Multiple Controllers by using Queuing Technique with queue monitoring variable**

**Initial Requirement:** CC is represented for Current Controller; SC is represented for Slave Controllers in the network;  $\lambda$  = Arrival Rate of the packet;  $\mu$  = Service Rate of the packet;  $f$  = Traffic Intensity of Controller;

Traffic Intensity ( $f$ ) of the controller act as a threshold value in the network.

**Result:**

0: No Need for Load Balancing

1: Successfully Load Balancing Done

**/\* Load Balancing between Multiple Controllers in SDN\*/**

if Load of CC >  $f$  then

{ for (i=0; i<n; i++)

{ /\* Calculate the traffic intensity value of all slave controllers SC<sub>i</sub> \*/

$$f_i = \lambda_i / \mu_i$$

Select controller SC<sub>i</sub> which has the lowest traffic intensity value in the network.

And monitor the length of the queue by using the queue monitoring variable.

} return 1;

} else

{ return 0; }

**Output:**

As a consequence, this algorithm helps to resolve all the issues which are related to load balancing in the network such as overloaded controllers and controller failure in the network. Moreover, also avoid cascaded failure of controllers in the network due to a load unbalancing between controllers.

Figure 5.8 (b): Algorithm for Load Balancing in Multiple Controllers



### 5.5.2 Numerical Evaluation and Result

Suppose the network (in Figure 5.9) has three controllers whose arrival and service rate are given in Table 12, by using the M/M/1: N/∞ Model in which the finite length of the queue is 10. Then calculate the parameters of these three controllers, which are stated in Table 13 and shown in Figures 5.10 to 5.12.

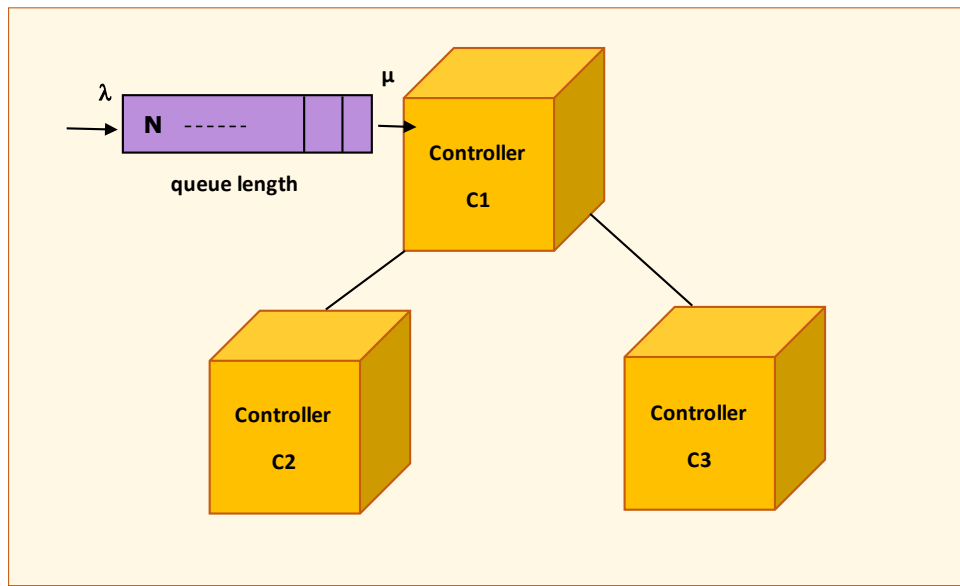


Figure 5.9: Scenario of Network using M/M/1: N/∞ Model for Load Balancing in SDN

Table 12: Arrival and Service Rate of the Controllers in the Network

Controllers	Arrival Rate	Service Rate
C1	8	9
C2	4	5
C3	3	6

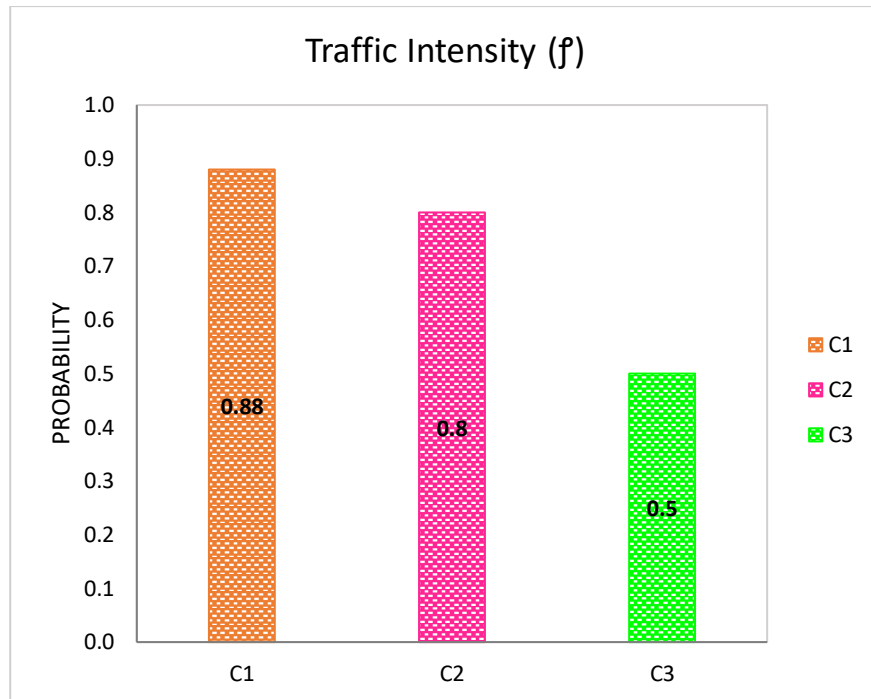


Figure 5.10: Probability of Traffic Intensity of the Controllers

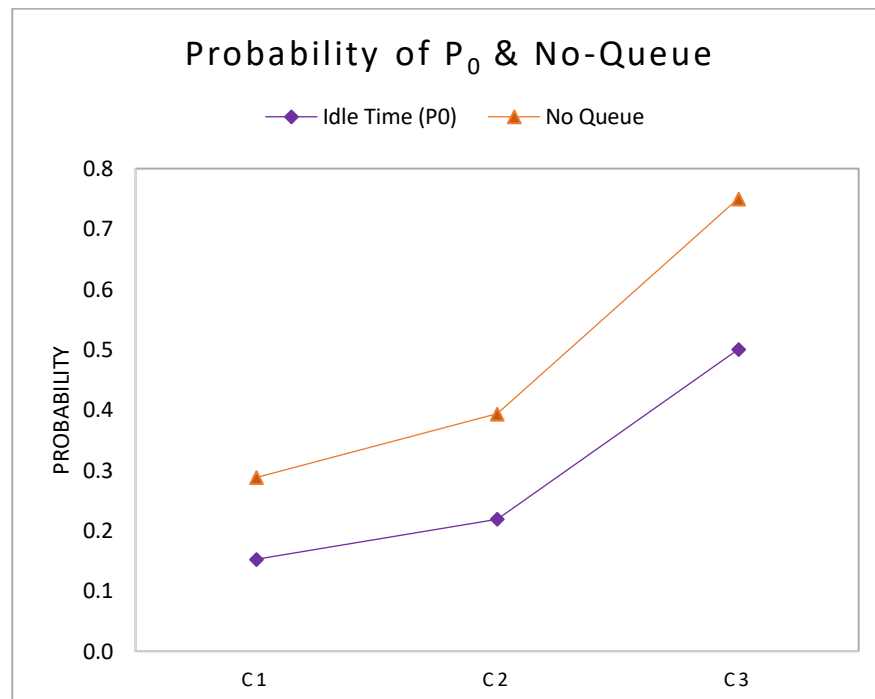
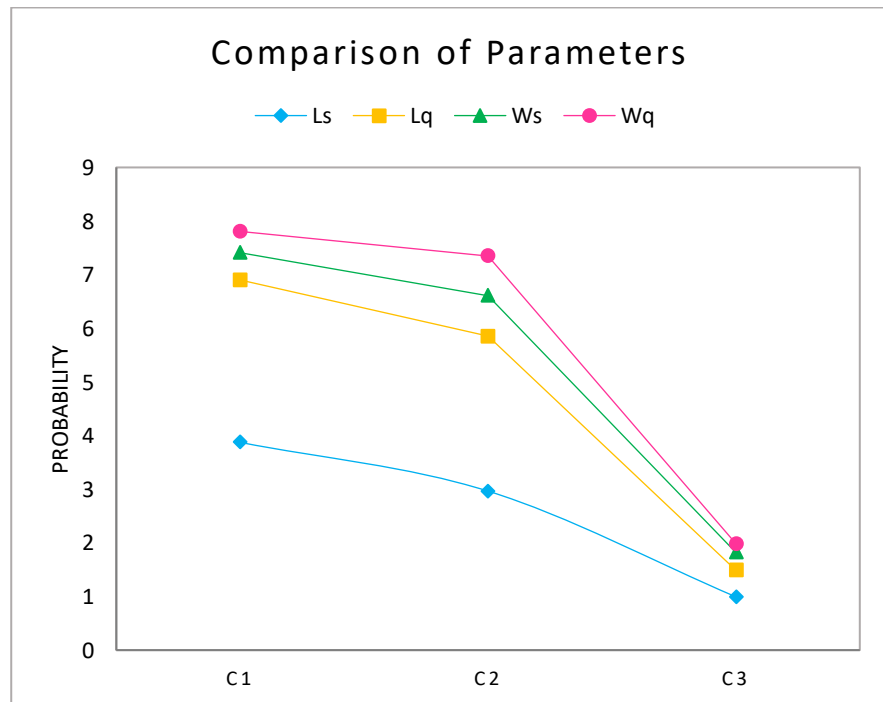


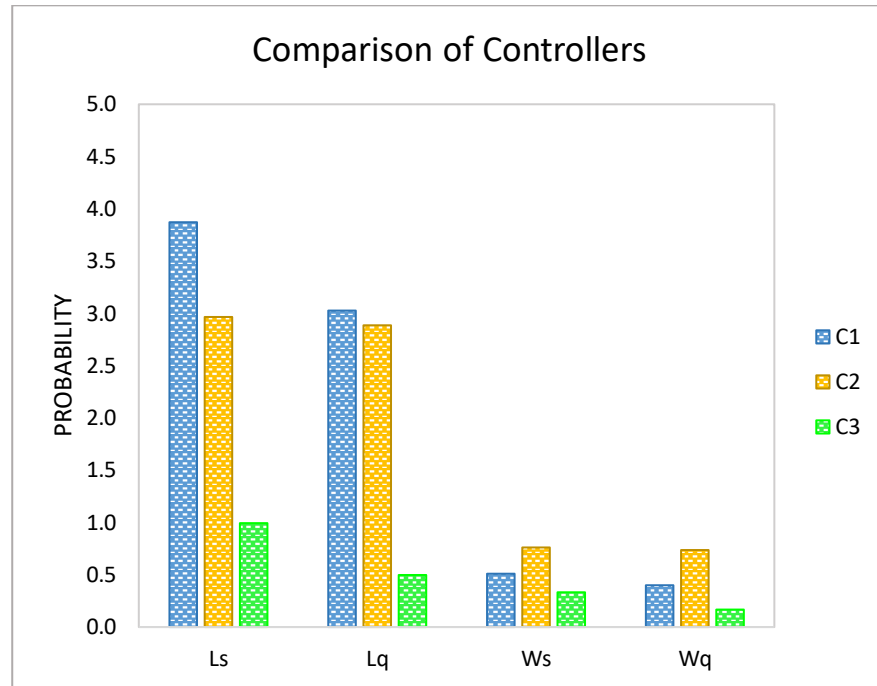
Figure 5.11: Probability w.r.t.  $P_0$  and No Queue occur in the System

Table 13: Comparison of Parameters of Controllers

Controllers	Traffic Intensity ( $\rho$ )	$P_0$	No Queue	$L_s$	$L_q$	$W_s$	$W_q$
C1	0.88%	0.12%	0.287%	3.87%	3.03%	0.508%	0.397%
C2	0.80%	0.20%	0.393%	2.96%	2.88%	0.759%	0.739%
C3	0.50%	0.50%	0.75%	0.99%	0.49%	0.332%	0.165%



(a)



(b)

Figure 5.12: Probability w.r.t.  $L_s$ ,  $L_q$ ,  $W_s$  &  $W_q$  occur in the System

In this scenario, controller C3 has the lowest traffic intensity value when compared to controller C2. whenever the threshold value of the network's present controller C1 is exceeded. The network load is managed by the controller C3.

## 5.6 Conclusion

The objective purpose is to design a load balancing algorithm by the integrated concept of Queuing Theory Technique and Markov Continuous Chain to manage the load balance between the SDN controllers, which reduces the packet dropping or packet loss ratio of the network. Lack of load balancing techniques causes a network imbalance. So, it is necessary to distribute the proper workload on controllers because network traffic fluctuates dynamically. Figure 5.7 highlights the average throughput, average delay, and packet delivery ratio in both models; If the queue size is reduced then dropped the rate of the packet is increased in the network.

But if the size of the queue is increasing or set at infinite then the packet delivery ratio increases but the average delay of a network is increasing rapidly. As a consequence, it increases the latency of the network, which is not tolerable for any communication. So, use the queue size more wisely for a network. Moreover, the comparison shows the M/M/1 finite capacity model is preferable to another model.

# **Chapter 6**

**Balancing through  
Probability Distribution  
in SDN**

# 6 CHAPTER

## **Balancing through Probability Distribution in SDN**

### **6.1 Introduction**

Software Defined Network provides an innovative paradigm of networking. It offers to increase programmability, and adaptability, enhance flexibility along with easy manageability, and dynamic reconfiguration of network elements. It provides help to fulfill the requirements of users and to improve network control that permits the network provider to respond to the changing business requirements. The Software Defined Network (SDN) is an upcoming technology in the networking design, which decouples the control plane from the data plane. The SDN controller receives all control logic and offers a centralized logical view of the complete network. But it also increases the chance of a failure in the network due to a single controller. To overcome this situation, multiple controllers are required in a network. When multiple controllers are used in SDN networks, there are problems with load balancing, such as when the controller is overloaded because the load is higher than its threshold value. This leads to controller failure or cascading failure of controllers, both of which are detrimental to the network. So, since it directly impacts network performance, distributing the workload among the various SDN controllers is one of the trickier responsibilities.

## 6.2 Related Work

D. Kreutz et al. [6] provide an exhaustive survey on Software Defined Network, although it provides an emerging paradigm of networking that gives hope to changes in the traditional network. Thus, it offers an opportunity to solve long-standing problems in a traditional network; because a controller has direct control over the network through well-defined application programming interfaces. The ongoing research and challenges in SDN are fault tolerance, load balancing of multiple controllers, scalability, synchronous, and so on. W. Braun et al. [8] address a lack of programming in the existing network. To overcome this situation by using software defined networking, the network can dynamically add new functionalities in the form of apps. The OpenFlow protocol supports a variety of message types that specify how a controller is synchronised with various network forwarding devices. The speed of a packet's processing affects the controller's estimation of the packet's overall sojourn time. Additionally, there is still work to be done on figuring out a controller's packet drop likelihood. Y. Yu et al. [15] give more attention to the reliability of networks which offers simplified network management and innovation in the networking field. To ensure the reliability of SDN via fault tolerance; but still, is in the initial stage. So, it is considered a future work for research in this direction. J. Chen et al. [18] suggested designing a mechanism for achieving fault tolerance in SDN which is useful for failure detection and recovery because it is unable to survive in a large-scale network while facing a failure;

Y. Zhang *et al.* [66] one of the main criticisms is a controller failure which decreases overall network performance and availability. Moreover, when multiple controllers are suggested but they increase the complexity of networks. At last highlights, potential research issues in multiple controllers in SDN such as coordination between controllers, load balancing among controllers, etc. A. Mahjoubi *et al.* [92] in SDN single controller suffer serious problems such as single point of failure, availability, and scalability. To overcome these problems, Distributed controllers are used, but they still have to deal with fault tolerance and load balancing challenges among controllers. I. F. Akyildiz *et al.* [24] SDN paradigm promotes innovation and evolution in networking which improves resource utilization, simplifies network



management, and decreases the operating cost of a network. The development of SDN architecture receives the majority of research attention, but traffic engineering receives very little. Traffic engineering must take into account this perspective in order for SDN to achieve higher scalability, availability, dependability, consistency, and precision. Examples include traffic analysis, load balancing, flow control, and fault tolerance.

B. Xiong *et al.* [75] the limitation of a logically centralized controller in SDN affects the network performance. The future efficiency of the controller could be approximately predicted using a queueing model. W. H. F. Aly *et al.* [89] an important aspect of resilience is fault tolerance which ensures the availability and reliability of a network is high. Both fault tolerance and load balancing are interrelated issues. Moreover, it manages the load between controllers by using performance metrics of the network.

M. K. Faraj *et al.* [104] load balancing strategy is required when congestion and overloading problem has occurred in the network. Then queue length is utilized for load balancing to reduce congestion in a network while using multiple controllers rather than a single controller. A. Mondal *et al.* [105] in this paper proposed a Markov chain-based analytical model in SDN that analysis the performance of packet flow through OpenFlow. Due to high delay, a significant percentage of packets either table-miss entry or output action are dropped from the network. In the future, it will be expanded with a queue system that helps to shorten the packet flow's latency.

### 6.3 Problem Formulation

The control plane and the data plane are dissociated by the agile model offered by the Software Defined Network (SDN). The ultimate authorities of SDN controller are to regulate the network traffic, insert appropriate flow rules in forwarding elements, maintain and update network topology, detect network failure and how to recover it, and so on. These responsibilities are handled by a controller because it provides a centralized logical view of the network; if it failed then the entire network

becomes halted. Thus, it is a complicated task for a single controller. Still, a single controller has a higher impact on failure in a network. But a central point of failure in the network can be avoided by multiple controllers. Meanwhile, some challenges are counter like uneven traffic distribution between controllers that become an origin of cascaded failure of controllers; when a controller manages network traffic beyond its capacity. Then drop rate of packets is increasing exponentially in a network. It is happening due to insufficient implementation of load balancing techniques in the network. As a consequence, some controllers are overloaded, and some are underloaded. When a controller has to handle network traffic more than its capacity or threshold value. Then the rate of packet loss is increase and the performance of the network is decreased. So, it is necessary to distribute the appropriate workload among the controllers.

## **6.4 Proposed Approach**

To propose an algorithm that gets rid of these challenges by evaluating an equilibrium state of distribution which describes the long-run probability of controllers by integrating Queuing Technique with Markov Continuous Chain, which aids in lessening the packet drop ratio and improving the efficiency of a network. Load Balancing is an imperative issue in multiple controllers for optimum utilization of resources in the network and minimum overhead in the control layer. To achieve load balancing between multiple controllers by distributing a load of the overloaded controller to other controllers of a network. Uneven load distribution among controllers reduces the overall utilisation of resources in the network [92,93]. As a consequence, some controllers are underloaded or idle in a network whereas some controllers reach their performance to bottleneck and upsurge response delay due to network traffic [88,89].

The controller is a crucial component in SDN because it manages all traffic of the network. A controller has the authority to make all routing decisions of a network. So, since it directly impacts network performance, balancing the workload across controllers is one of the trickiest tasks. Additionally, the performance of the network

is considerably affected by the control plane's scalability and load balancing between them.

The motive of the proposed approach is how to maintain and control the load balancing between the multiple SDN controllers. Both fault tolerance and load balancing are complicated/complex and interrelated issues when dealing with multi controllers in a network. To design an algorithm by the integrated concept of Queuing Theory Technique and Markov Continuous Chain to manage the load balance between the SDN controllers, which reduces the packet dropping or packet lost ratio of a network. It is happening in a network due to the lack of load balancing techniques. So, it is necessary to distribute the proper workload on controllers because network traffic fluctuates dynamically.

Network traffic behaves like a stochastic process whose behavior is random changes over time. The Markov process is a simple stochastic process, in which the distribution of the future state depends only on the present state of the process rather than on how to arrive at the present state. Thus, the stochastic process must be holding the Markov property “memoryless” [74,100].

According to this property, the probability of the future state ( $Y_{t+1}$ ) at time instant ( $t+1$ ) depends upon the present state ( $Y_t$ ) at time instant ( $t$ ) is outlined in equation 6.1 as:

$$P [ Y_{t+1} | Y_t, Y_{t-1}, \dots, Y_2, Y_1, Y_0 ] = P [ Y_{t+1} | Y_t ] \quad (6.1)$$

Where,  $Y_{t+1}$  dependent on  $Y_t$  but not dependent on  $Y_{t-1}, Y_{t-2}, \dots, Y_2, Y_1, Y_0$ .

The Mathematical Notation of Markov Property as follow:

$$P (Y_{t+1} = A | Y_t = A_t, Y_{t-1} = A_{t-1}, \dots, Y_0 = A_0) = P (Y_{t+1} = A | Y_t = A_t)$$

for all  $t = 1, 2, 3, \dots$  and for states  $A_0, A_1, \dots, A_t, A$

The memoryless property is also applicable to the queuing model. It is often referred to as the network queuing model. As shown in Figure 6.1, there are two different kinds of events that could occur in a queuing model: either the packet's arrival rate

or service rate. The poisson distribution and the exponential distribution are followed, respectively, by the arrival rate, the time between arrivals, and the service rate [74,100].

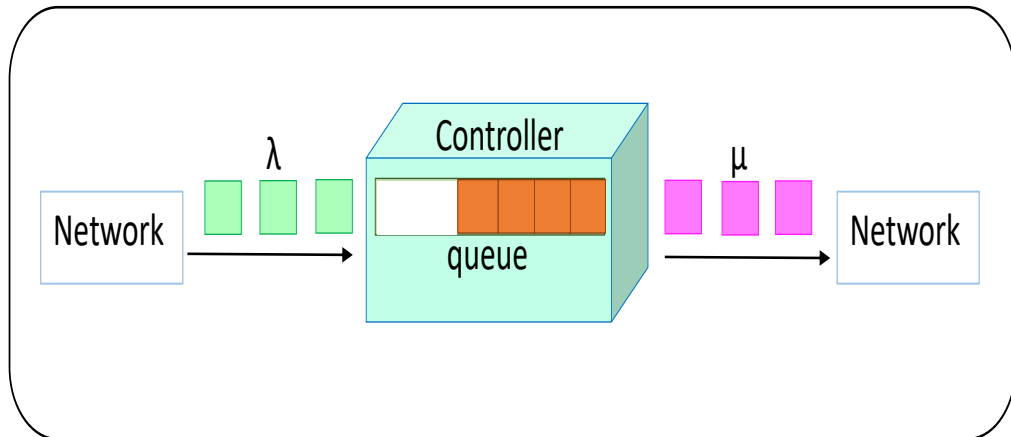


Figure 6.1: Use of Queuing Technique in SDN Controller

#### 6.4.1 Markov Chain

The Markov chain is a vital and significant tool for analyzing a transition matrix; describing the probability of all possible states in a transition diagram. A transition matrix must be a square (same number of rows and columns) matrix and represented by  $P$ . The sum of the probability of each row of the transition matrix is equal to one. In general, the probability of the next state is depending on the previous state; probability from one state to another state is represented in equation 6.2 as:

$$p_{ij} = P(X_{t+1} = j | X_t = i) \quad (6.2)$$

It is also known as the conditional probability of the state [94]. This specified that the probability of the next state “j” is given by the probability of the present state “i”. The Markov chain's transition matrix is denoted by the symbols  $P$  or  $(p_{ij})$ .

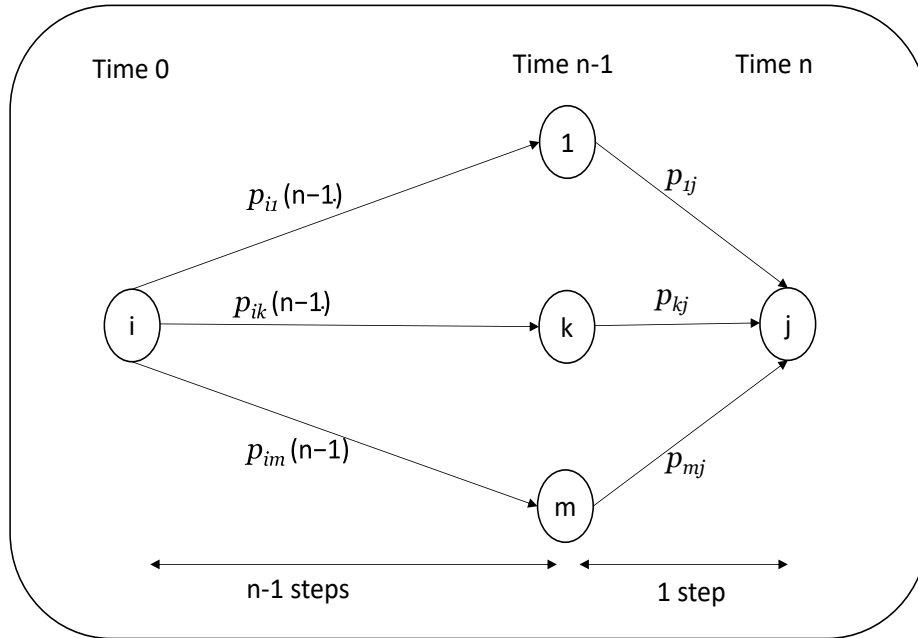
#### 6.4.2 Transition Probability of ‘n’ Steps

Let suppose Markov Chain in a space  $S$  is  $\{X_0, X_1, X_2, X_3, \dots\}$ , with size  $N$ . The transition probabilities for the Markov Chain is outlined as:

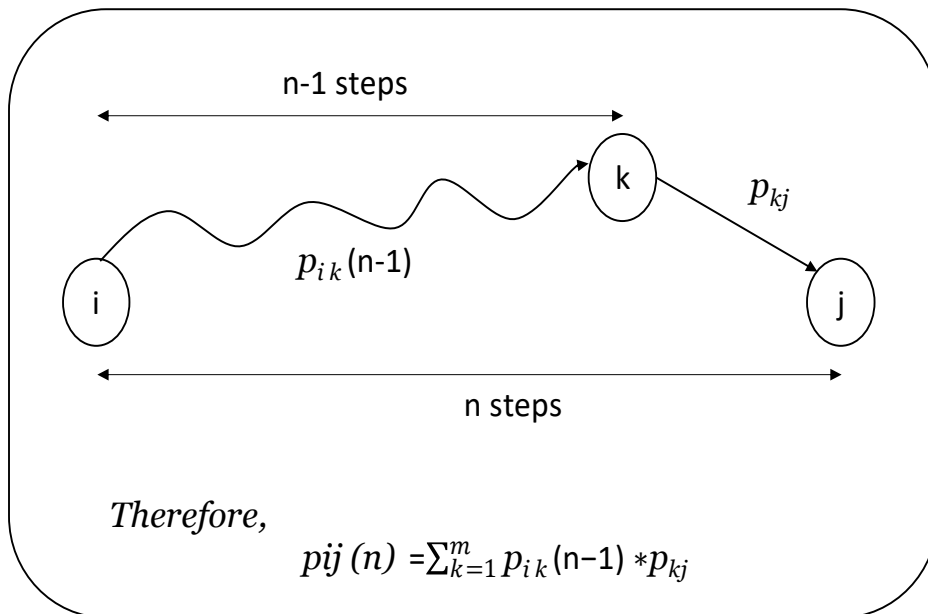
$$p_{ij} = P(X_{t+1} = j | X_t = i)$$

for  $i, j \in S, \quad t=0,1,2, \dots$

$$p^{(n)}_{ij} = P(X_{n+1} = j | X_1 = i)$$



(a)



(b)

Figure 6.2: Transition Probability of 'n' step time period

Then, Transition Probability from i state to j state after one step time period as given below in equation 6.3:

$$p_{ij} = P(X_{n+1} = j | X_n = i) \quad (6.3)$$

Transition Probability from i state to j state after “n” step time period in Figure 6.2 (a and b) respectively.

### 6.4.3 Equilibrium State of Probability

Let  $P_{ij}$  = conditional probability of “j” state is given by the current state “i”. The initial probability of a state is represented by  $\pi_0$ . The size of the vector state of probabilities is  $1 \times N$ . The vector state of probabilities is used to determine the probability that the system is in this state. This information is placed into a vector state of probabilities such as:

$$\begin{aligned} \pi(k) &= \text{vector state of probabilities for period } k \\ &= (\pi_1, \pi_2, \pi_3, \dots, \pi_n) \end{aligned}$$

where as

$$n = \text{number of states}$$

$\pi_1, \pi_2, \pi_3, \dots, \pi_n$  = probability of being in state 1, state 2, ..., state n.

Furthermore, if computing the state probability for period n+1 by using the state probability of period n.

$$\pi(\text{next period}) = \pi(\text{current period}) * P$$

or

$$\pi(n+1) = \pi(n) * P$$

and the sum of state distribution must be one

$$\pi_1 + \pi_2 + \dots + \pi_n = 1$$

A Markov Chain provides a facility to evaluate the steady-state (or equilibrium state) of distribution. A steady-state probability is an irreducible set of states which is represented by the  $\pi_j$ . Let P be the transition matrix for the ‘n’ state. According to the steady-state theorem or equilibrium behavior of the Markov Chain is:

$$\lim_{n \rightarrow \infty} (p_{ij})^n = \pi_j$$

The steady-state distribution is  $[\pi_1 + \pi_2 + \dots + \pi_n = 1]$

$$\pi_{(n+1)} = \pi_{(n)} * P$$

At equilibrium, it is known that

$$\pi_{(n+1)} = \pi_{(n)}$$

Therefore, 
$$\pi = \pi P \tag{6.4}$$

Thus, an equilibrium state of distribution is used to describe the long-run behavior (or probability) of a state or process. At the equilibrium state, the probability of the next period is the same (equal/identical) as a state of the probability of the current period (in equation 6.4). Thus, the initial state of probability values does not influence the equilibrium state of probability. This concept is applied to calculate the probability of controllers in a network domain.

#### 6.4.4 Equilibrium State in a Queue

How an equilibrium state is implemented in a queue is shown in Figure 6.3; The state diagram shows the probability of queue states. If a queue is empty then the probability is “1-p”. Similarly, if queue length reaches “n state” then the probability is “1-q”. The probability of one state (state 1) to another state (state 2) is “p”. The probability or likelihood of transitioning from state 2 to state 1 is denoted by q," and "1-p-q" denotes the likelihood of remaining in the same state (such as in a self-loop); Always the sum of the probability of transition is one.

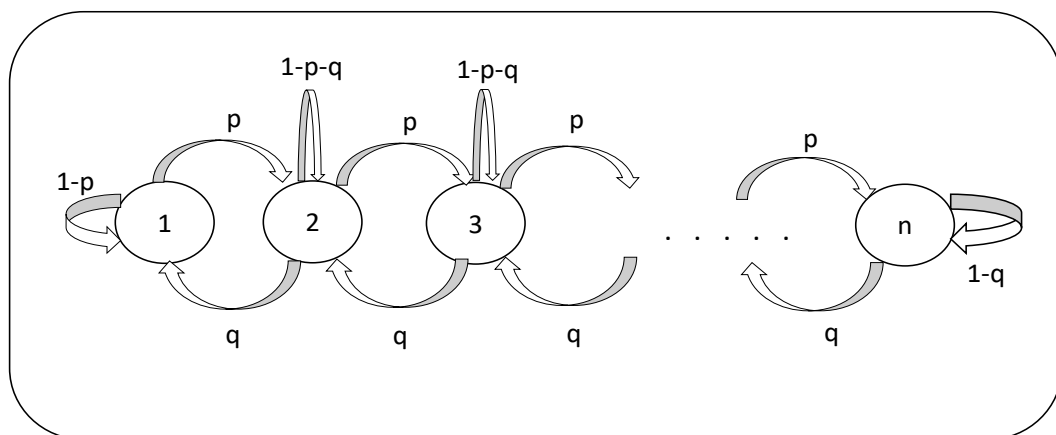


Figure 6.3: Equilibrium State in Queue

### 6.4.5 Pseudo Code for Load Balancing in SDN Controllers by using the Queuing Technique with the Markov Chain

For designing an adaptive load balancing algorithm, multiple controllers are used. The master controller has full control of the network. When the length of the queue (incoming to the master controller) exceeds the specified queue size of the controller, a slave controller, with the highest probability among active slave controllers in the network, is selected. According to the Markov Chain, the future state of a network only depends upon the present state of a network rather than how arrived in the present state ( $p^{(n)}_{ij} = P(X_n = j | X_{n-1} = i)$ ). The steady-state distribution or equilibrium state of a Markov Chain is used to calculate the probability of controllers in the network domain, whose summation ( $\Sigma$ ) of probabilities is always equal to one [100]. Table 14 shows the abbreviation of variables used in the algorithm; Figure 6.4 shows the pseudo-code for load balancing in multiple controllers.

Table 14: Abbreviation of Variables used in Algorithm

Abbreviation of Variables	
Variable Name	Purpose
th	Threshold Value of Controller
lc	Load of Current Controller
ql	Queue Length
qs	Queue Size

#### ***/\* Pseudo Code for Load Balancing for Multiple Controllers in SDN by using the Queuing Technique with the Markov Chain \*/***

##### ***Initial Requirement:***

$\pi_0$  has represented the initial probability of controllers, which is a  $1*N$  size matrix;  
 $P_{ij}$  has represented the Transition Probability of Matrix  $P$ ; it is a square matrix  $N*N$ ;  
 $i$  and  $j$  represent row and column position in the transition probability matrix  $P$ ;

##### **Step 1:**

$M=1$  /\* Initial Probability of Master Controller\*/

$S=0$  /\* Initial Probability of all Salve Controller\*/



```
 $\pi_0$  /* Initial Probability of Controllers whose size is a 1*N matrix */  
P=[i][j] /* Transition Probability of Matrix P which is a square matrix N*N */
```

**Step 2:**

```
for each controller in network  
do loop  
if ( $\pi_{i+1} < \pi_i P$ )  
then  
/*Calculate the Steady-State Probability of Distribution by using the formula  
 $\pi_{i+1} = \pi_i * P$  */  
exec proc steady_state;  
end if;  
end loop;
```

**Step 3:**

```
/* Once's steady-state probability is reached; it becomes independent of the  
initial probability of controllers */
```

```
th=probability distribution of controller act as a threshold value;  
lc=current load of controller;  
if (lc < th and ql < qs)  
then  
/*All the computations of the network are controlled by the Master  
Controller*/  
exec proc master-controller;  
else
```

```
/* Retrieve the Slave Controller which has a higher probability as
compared to the other slave controllers in the network domain and its
Queue_Length is less than specified Queue_Size */

    exec proc slave-controller;

end if;

end;
```

**Output:**

Therefore, the distribution of workload among controllers depends upon the probability of distribution controllers; which aids to lessen the threat of controller failure as well as managing load balancing between controllers.

Figure 6.4: Pseudo Code for Load Balancing in SDN Controllers

## 6.5 Simulation and Evaluation of Result

A centralized controller manages all responsibilities of the network, then it come very difficult to accomplish their responsibilities simultaneously. It is a very important characteristic of networking to monitor the network traffic in real-time. For this purpose, queuing technique that effectively uses the global perspective or view of the network to control the congestion of the network is proposed. Moreover, it also provides the facility to determine packet delivery ratio, and packet drop ratio along with the various queue objects. Thus, to determine what is the significance of queue size to control packet drop rate in the network. The quality-of-services (QoS) of the network is enhanced by these parameters.

### 6.5.1 Queue Model

Using queue model M/M/1 with unlimited (or infinite) and limited (or finite) capacity, all required manipulations were carried out on both queue models using simulator, as illustrated in Figures 6.5. (a and b).

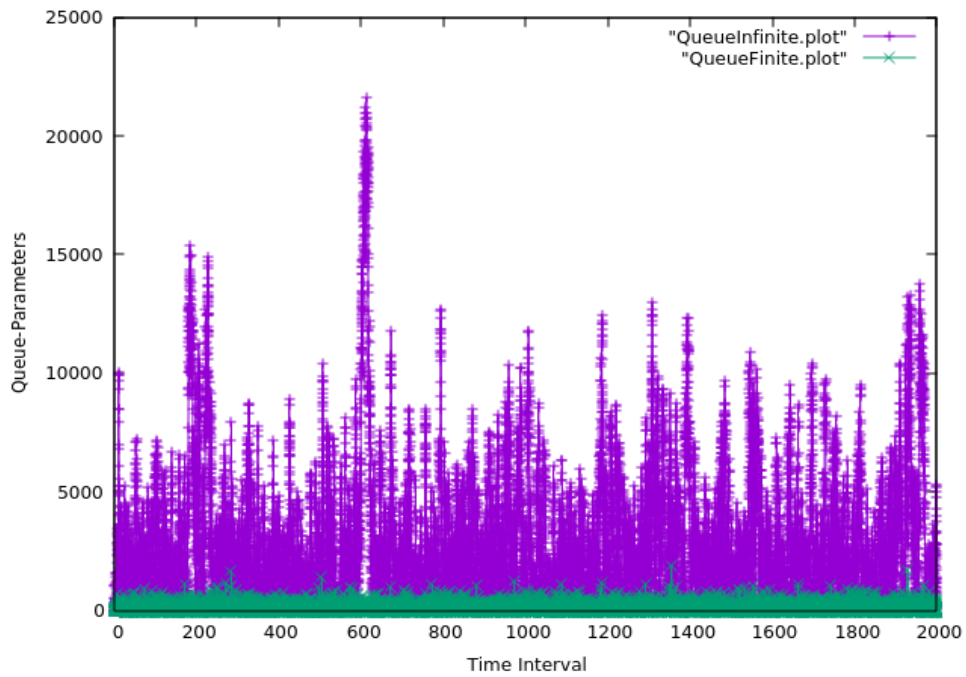


Figure 6.5 (a): Comparison between both Queue Models

\*Note: “Y” axis represents Queue-Parameters like  $qsizeB$ ,  $qsizeP$ ,  $arrivedP$ ,  $departedP$ ,  $droppedP$ ,  $arrivedB$ ,  $departedB$  and  $droppedB$ ; where \*B is number in Bytes, and \*P is number in Packets

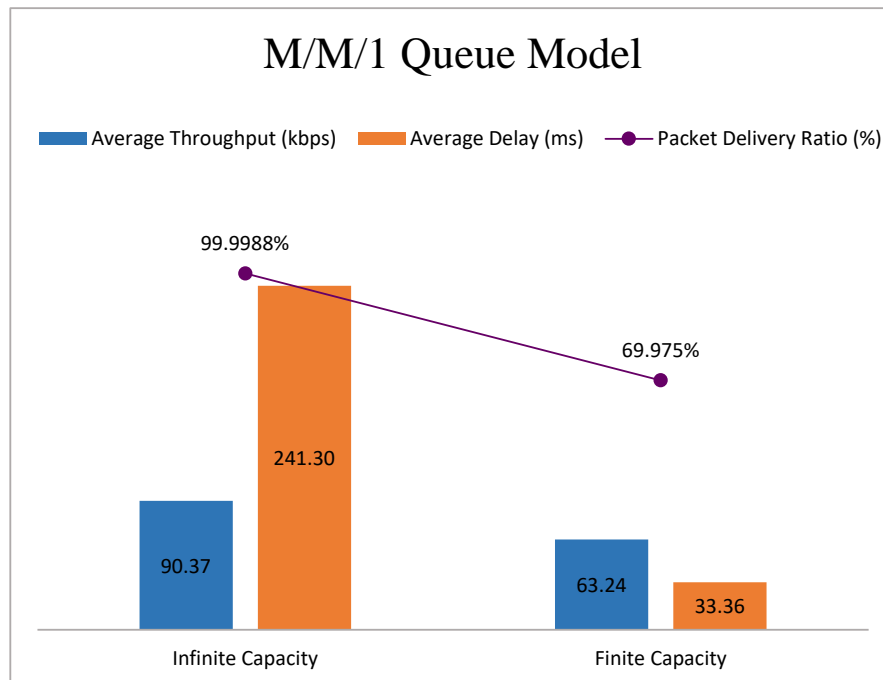


Figure 6.5 (b): Comparison between both Queue Model in Average Throughput, Average Delay and Packet Delivery Ratio

Figure 6.5 (b) highlights the average throughput, average delay, and packet delivery ratio in both models, where the blue, orange, and purple color represents respectively. The average delay of the unlimited queue model is much higher than the limited queue model; because when the size of the queue is increasing or set to infinite then the packet delivery ratio increases, but at the cost of an increased average delay, which is not affordable or acceptable for any communication. If the size of the queue is set to finite, then the average delay of the network becomes lessened, as shown in Figure 6.5 (b). Based on these factors, the model with a finite capacity queue is better than the model with an infinite queue.

### 6.5.2 Various Queue Objects

Many different types of queue objects are available like DropTail, Stochastic Fair Queue, and so on. These queue objects show varying behavior in terms of the probability of packet delivery ratio and packet drop ratio in Figure 6.6 (a to c).

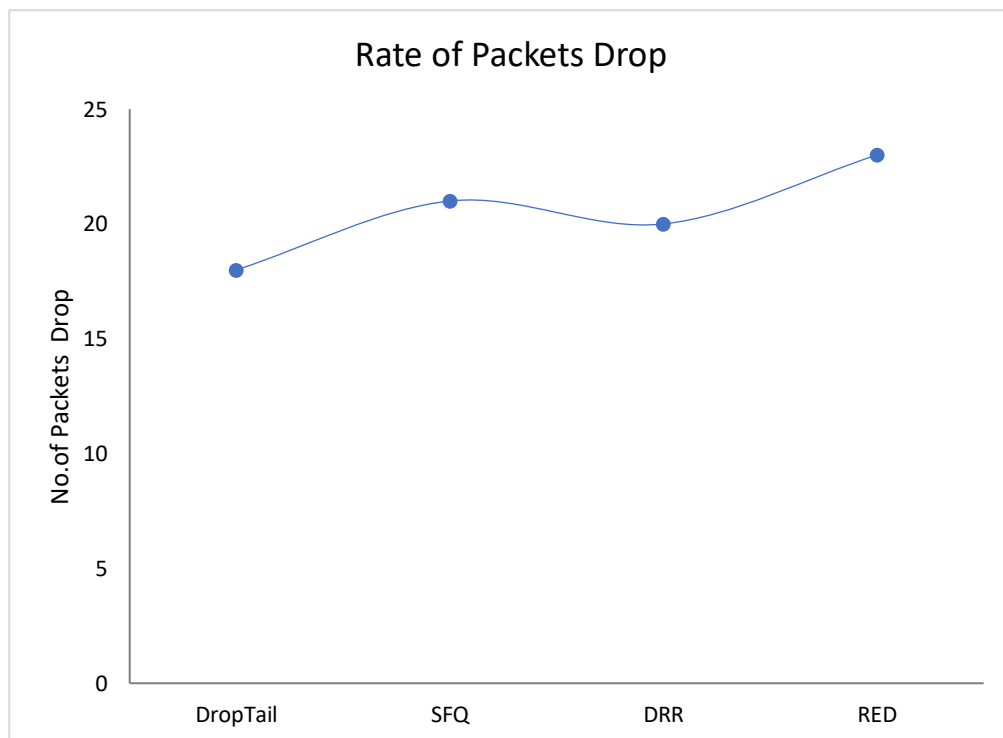


Figure 6.6 (a): Number of Packets Drop in various Queue Object

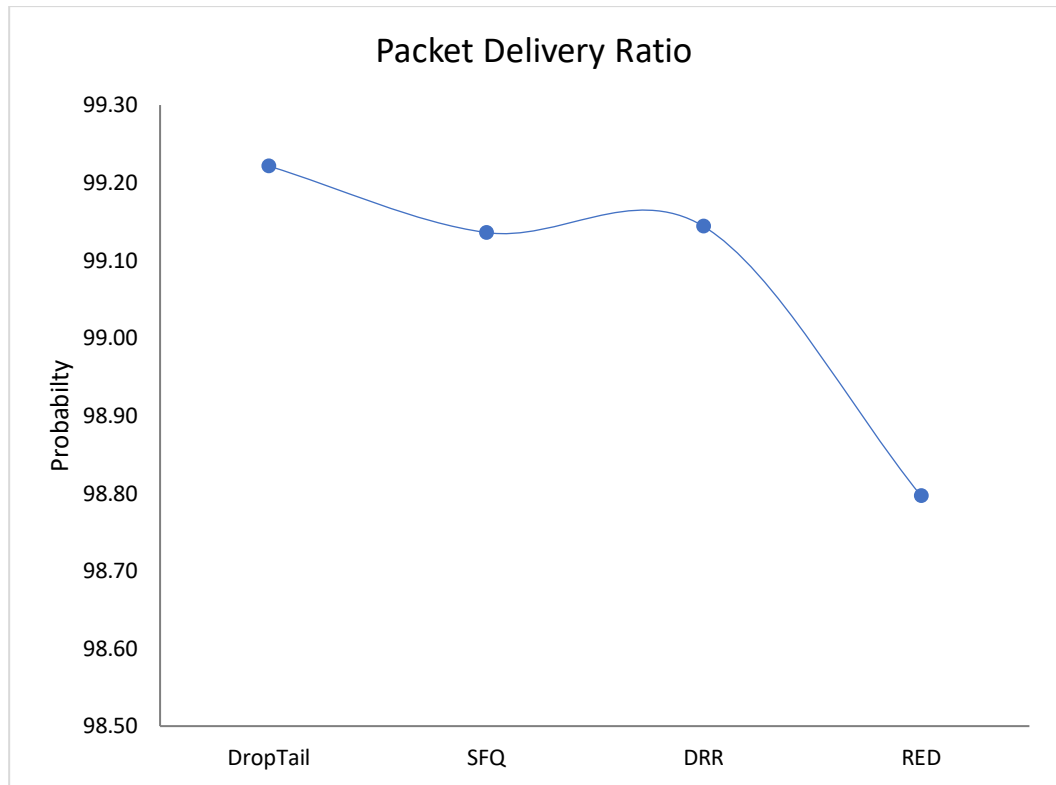


Figure 6.6 (b): Probability of Packet Delivery Ratio in various Queue Object

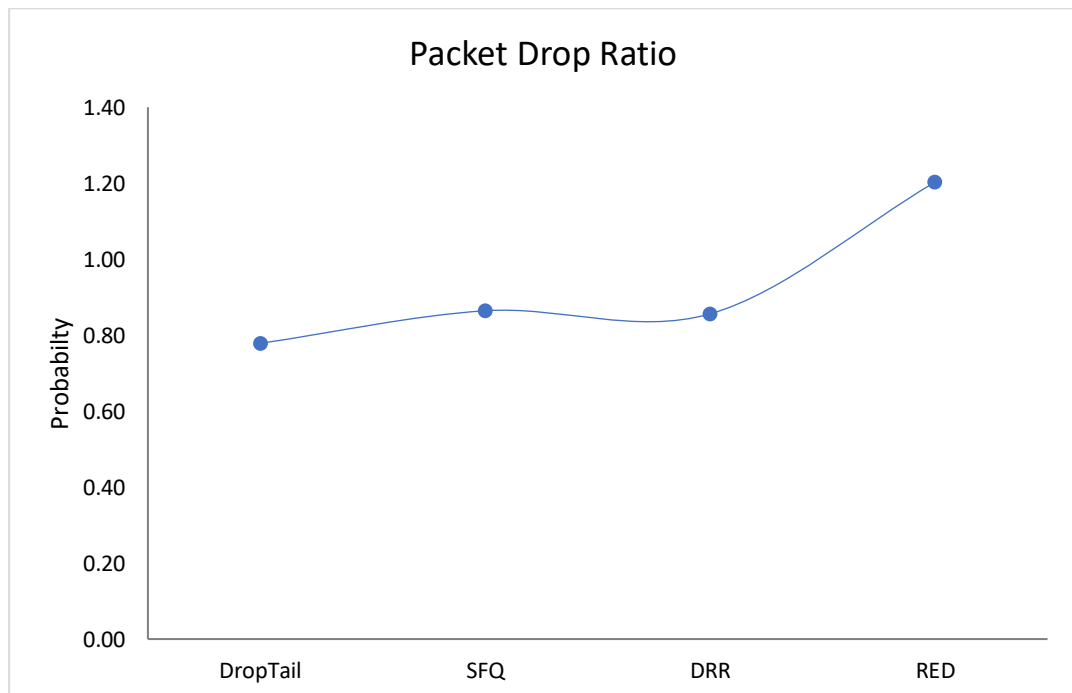


Figure 6.6 (c): Probability of Packet Drop Ratio in various Queue Object

The DropTail queue object shows a packet delivery ratio of 99.22% which is higher than other objects and a packet drop ratio of 0.78% approximately which is lower than another queue object. To select finite capacity queue model with DropTail queue object after analyzing the simulation result.

### 6.5.3 Equilibrium State of Controllers

Take another scenario of two controllers in which the value of transition matrix (P) is sequentially increased in every case and calculate the steady-state of probability distribution up to sixteen steps respectively. Let me explain a case, and how to evaluate an equilibrium state of controllers in this scenario. If an initial probability of both controllers at time period zero is  $\pi_0 = [1 \ 0]$ . The matrix of transition of probabilities for the controller is  $P = \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix}$  (as shown in Figure 6.7), where M state the representation of the master controller and S state for slave controller.

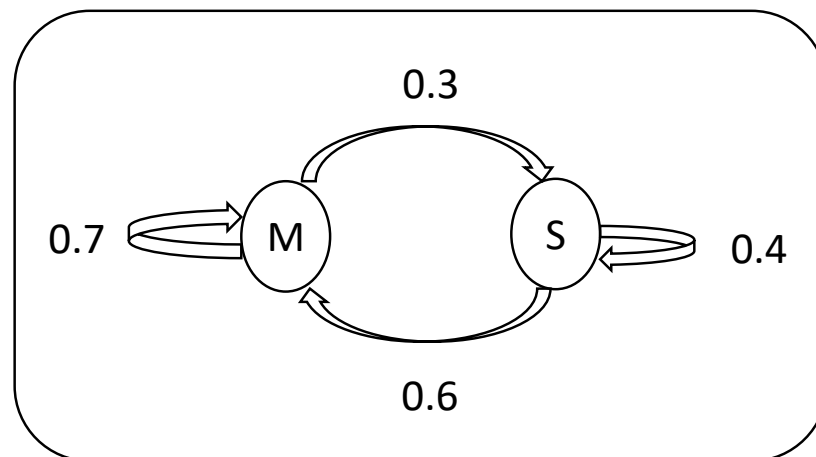


Figure 6.7: Graphically Representation of Controllers

where as

$P_{11} = 0.7$  probability of the master controller will be functioning for handling the network traffic or workload.

$P_{12} = 0.3$  probability of slave controller will be functioning for handling the network traffic or workload.

$P_{21} = 0.6$  probability of the master controller will be functioning for handling the network traffic or workload.

$P_{22} = 0.4$  probability of slave controller will be functioning for handling the network traffic or workload.

Because these conditions/events are mutually exclusive and exhaustive as a whole, the total of the row probabilities must be one. When the current period is 0 then compute the state of probabilities for period 1 by using this formula:

$$\pi_1 = \pi_0 * P$$

$$\pi_1 = [1 \quad 0] \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix}$$

$$\pi_1 = [0.7 \quad 0.3]$$

Then calculate the probability of controllers in various steps until the value of  $\pi_{n+1} = \pi_n * P$ ; as shown in Table 15 and Figure 6.8. An equilibrium state of the controller is observed in Step 6 (Table 15), which is highlighted in yellow color. A few cases are shown graphically in Figures 6.9 (a to c) respectively.

Table 15: Equilibrium State for Two Controllers

No. of Steps	Two Controllers	
	Master	Slave
Step 1	1.000	0.000
Step 2	0.700	0.300
Step 3	0.640	0.360
Step 4	0.628	0.372
Step 5	0.626	0.374
<b>Step 6</b>	<b>0.625</b>	<b>0.375</b>
Step 7	0.625	0.375
Step 8	0.625	0.375

Step 9	0.625	0.375
Step 10	0.625	0.375
Step 11	0.625	0.375
Step 12	0.625	0.375
Step 13	0.625	0.375
Step 14	0.625	0.375
Step 15	0.625	0.375
Step 16	0.625	0.375

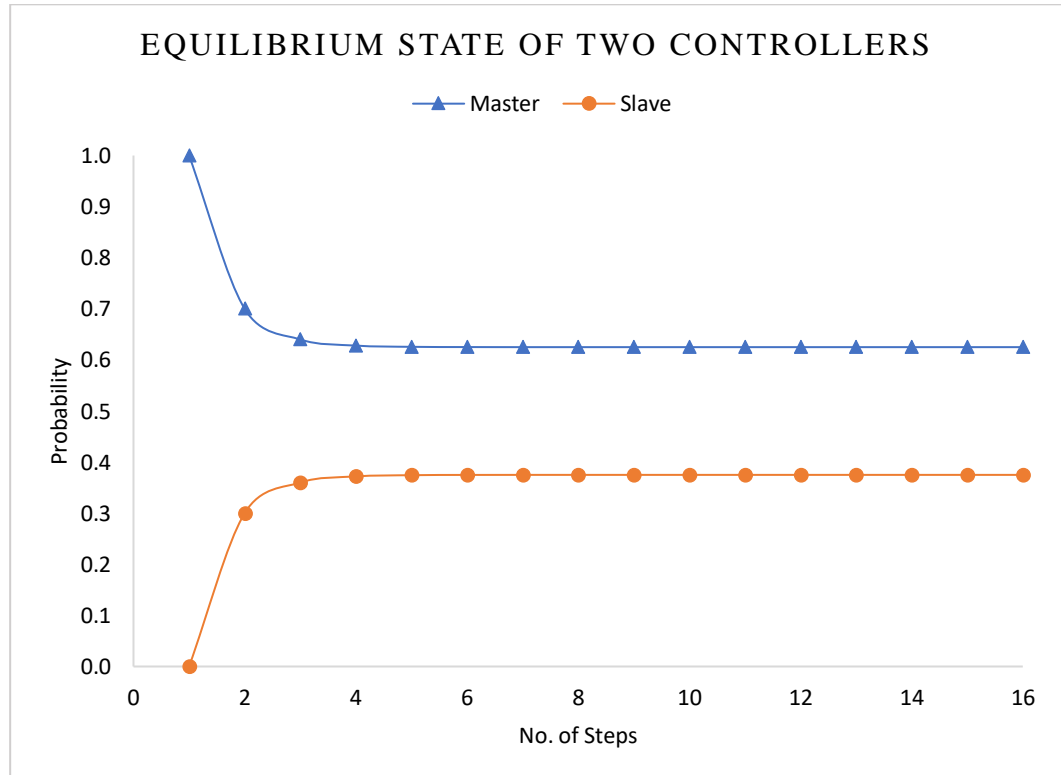
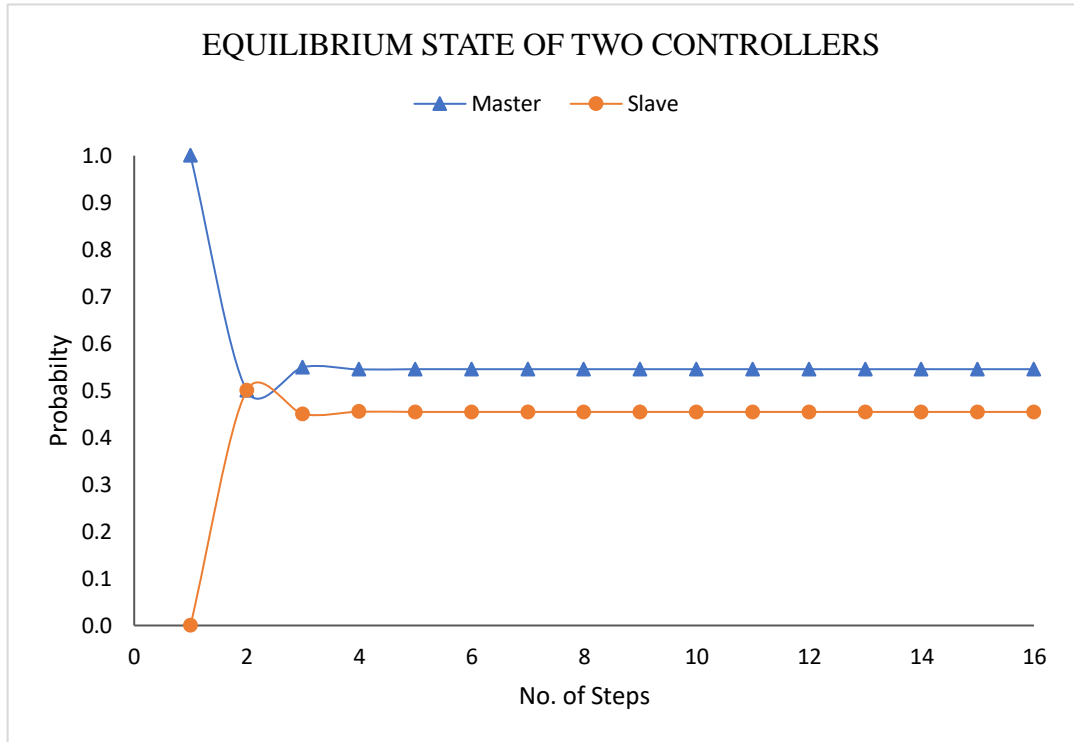
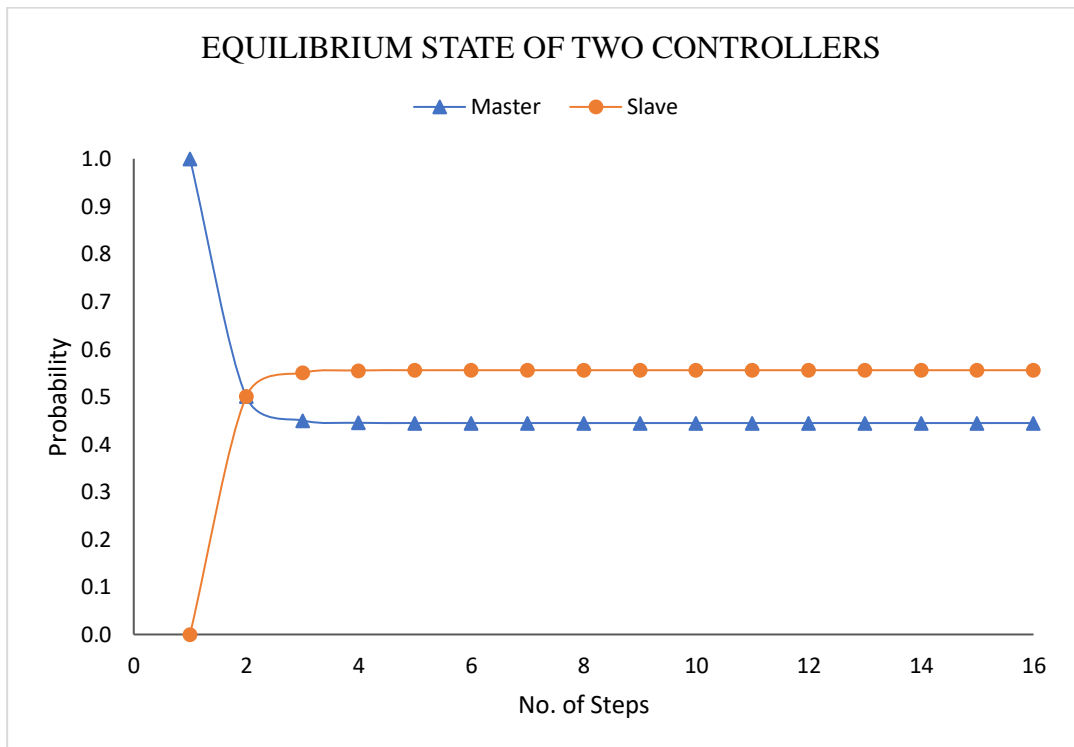


Figure 6.8: Equilibrium State of Two Controllers

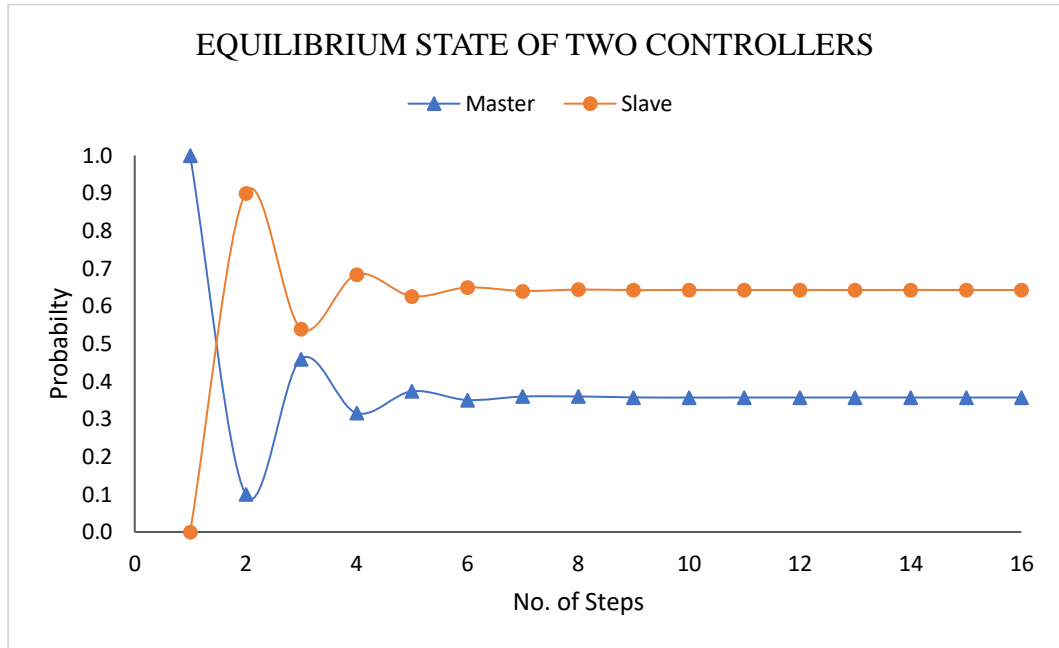




(a)



(b)



(c)

Figure 6.9: A few cases of Equilibrium State of Two Controllers

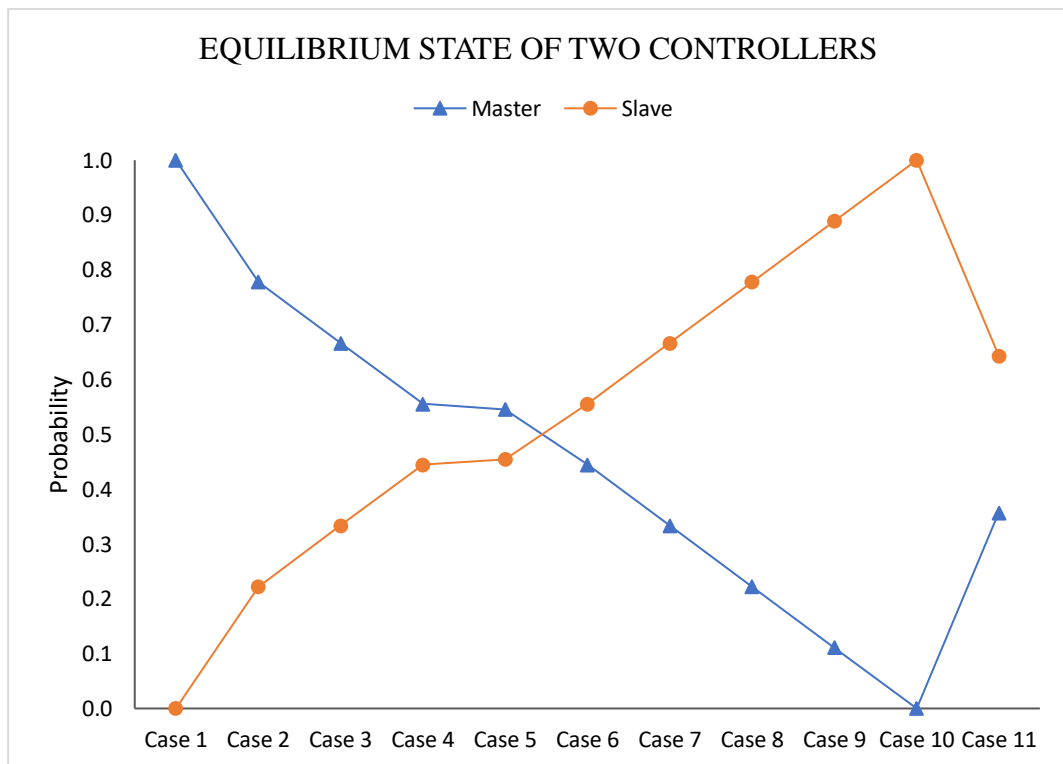


Figure 6.10: Graphically Representation of an Equilibrium State of Controllers in Different Cases

Table 16: Equilibrium State of Two Controllers in Different Cases

Cases	1	2	3	4	5	6	7	8	9	10	11
Master	1	0.777778	0.666667	0.555556	0.545455	0.444445	0.333334	0.222223	0.111111	0.000001	0.357142
Slave	0	0.222222	0.333333	0.444444	0.454545	0.555555	0.666666	0.777777	0.888889	0.999999	0.642858

Figure 6.10 and Table 16 show the analysis when the probability of a master controller is decreased from 100 to 0 percent and vice versa. Likewise, for the three controllers' scenarios, observations are recorded in Table 17 and Figure 6.11, and an equilibrium state observation is in Step 9 (in Table 17), which is highlighted with yellow color. Similarly, in the case of four controllers, observations are recorded in Table 18 and Figure 6.12, and an equilibrium state observation can be obtained in Step 10 (in Table 18), which is highlighted in yellow color. These findings lead to the conclusion that the complexity of the network grows along with the number of controllers. Using this strategy, it becomes easy to manage the congestion of a network and minimize the chance of controller failure in the network due to an imbalance of workload. This strategy also reduces the overheads induced during the switch migration, and context switching and minimizes the maintenance cost of the network.

Table 17: Equilibrium State for Three Controllers

No. of Steps	Three Controllers		
	Master	Slave 1	Slave 2
1	1.0000	0.0000	0.0000
2	0.8000	0.1000	0.1000
3	0.7500	0.1200	0.1300
4	0.7370	0.1250	0.1380
5	0.7336	0.1263	0.1401
6	0.7327	0.1266	0.1407
7	0.7327	0.1266	0.1407
8	0.7325	0.1267	0.1408
9	0.7324	0.1268	0.1408
10	0.7324	0.1268	0.1408
11	0.7324	0.1268	0.1408
12	0.7324	0.1268	0.1408
13	0.7324	0.1268	0.1408
14	0.7324	0.1268	0.1408
15	0.7324	0.1268	0.1408

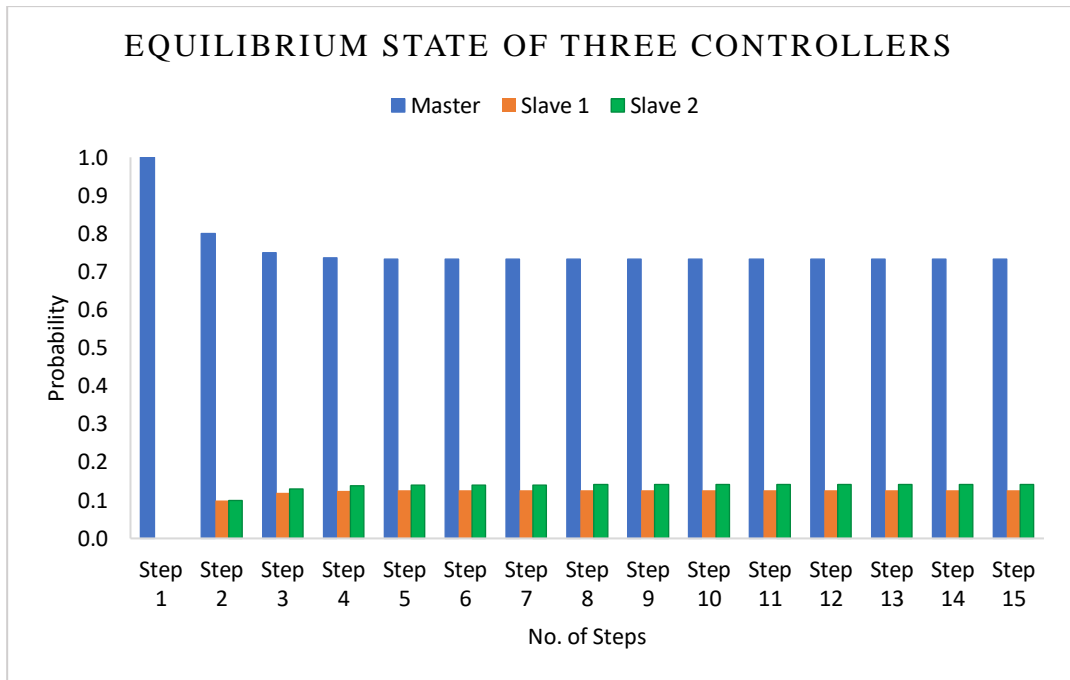


Figure 6.11: Equilibrium State of Three Controllers

Table 18: Equilibrium State for Four Controllers

No. of Steps	Four Controllers			
	Master	Slave 1	Slave 2	Slave 3
1	1.0000	0.0000	0.0000	0.0000
2	0.7	0.11	0.1	0.09
3	0.631	0.1345	0.129	0.1055
4	0.61565	0.139935	0.13635	0.108065
5	0.612324	0.14114	0.138077	0.10846
6	0.611619	0.141408	0.138461	0.108512
7	0.611473	0.141468	0.138543	0.108516
8	0.611443	0.141482	0.13856	0.108515
9	0.611437	0.141485	0.138564	0.108514
10	0.611436	0.141486	0.138564	0.108514
11	0.611436	0.141486	0.138564	0.108514
12	0.611436	0.141486	0.138564	0.108514
13	0.611436	0.141486	0.138564	0.108514
14	0.611436	0.141486	0.138564	0.108514
15	0.611436	0.141486	0.138564	0.108514

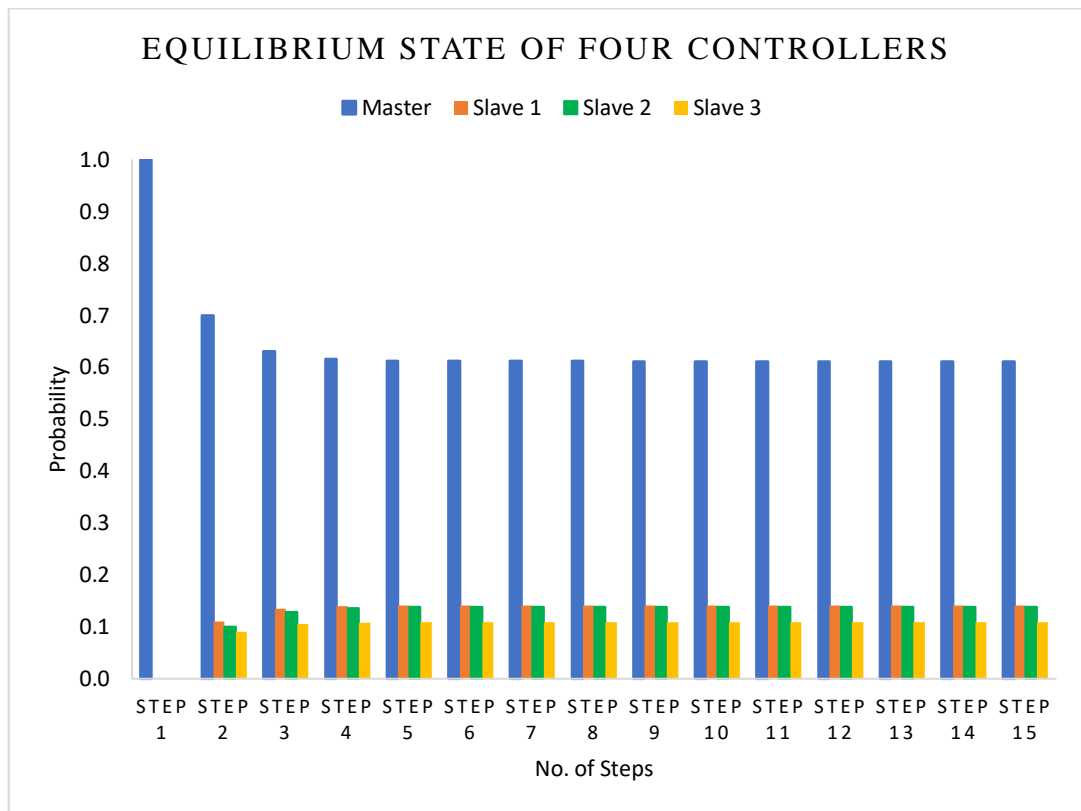


Figure 6.12: Equilibrium State of Four Controllers

#### 6.5.4 Significance of Queue Size

Next, examine toward what happens to the packet drop rate as the queue size is sequentially increased. As the size of the queue is reduced, the network's dropped packet rate increases. But if the size of the queue is increased by or set to infinite then the packet delivery ratio is improved, but on the other hand average delay of the network increases rapidly. Due to the increased network latency, which is not tolerable and unaffordable for any communication, the sizing of the queue needs to be done more wisely for the network. Figure 6.13 illustrates how the rate of packet drops decreases as queue size increases. As a result, as shown in Figures 6.13 to 6.15, the packet delivery rates increase and the packet drop ratio decreases as the queue size increases (from 5 to 50 in Table 19) respectively.

Table 19: Effect of Queue Size w.r.t. other Parameters

Queue Size	Packet Drop	Packet Delivery Ratio	Packet Drop Ratio
5	40	98.17	1.83
10	18	99.22	0.78
15	14	99.37	0.63
20	16	99.33	0.67
25	9	99.62	0.38
30	6	99.75	0.25
35	3	99.87	0.13
40	0	100.00	0.00
45	0	100.00	0.00
50	0	100.00	0.00

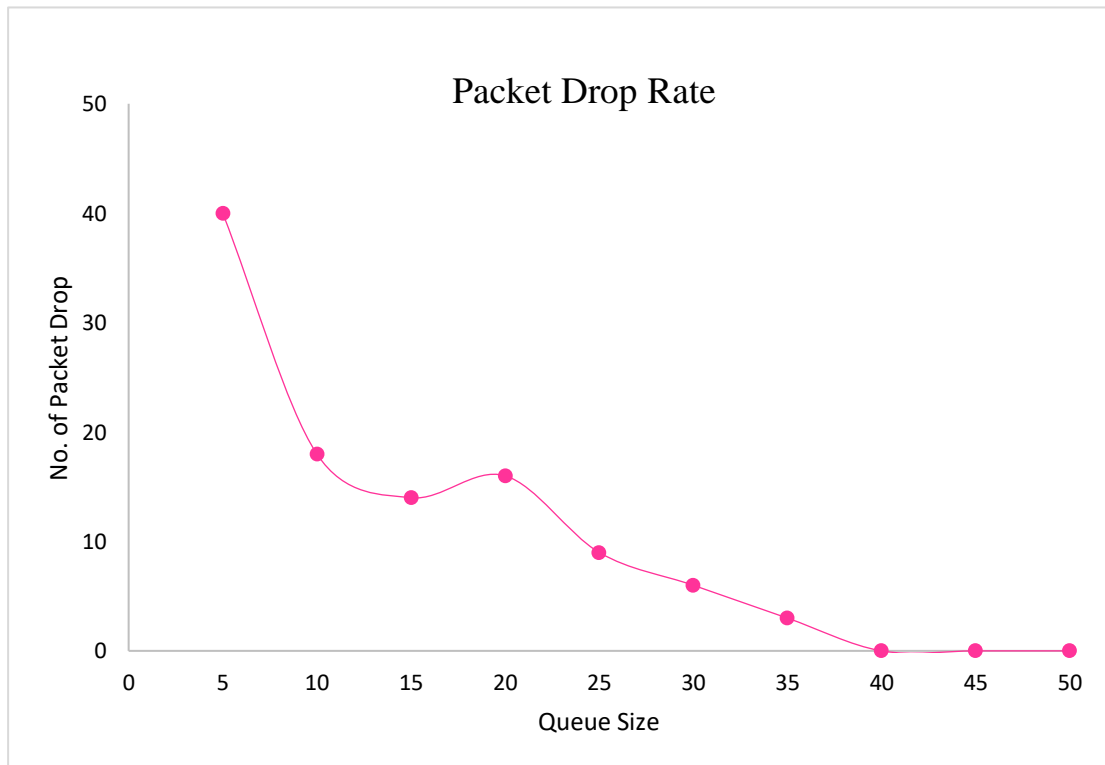


Figure 6.13: Drop Rate of Packets v/s Queue Size

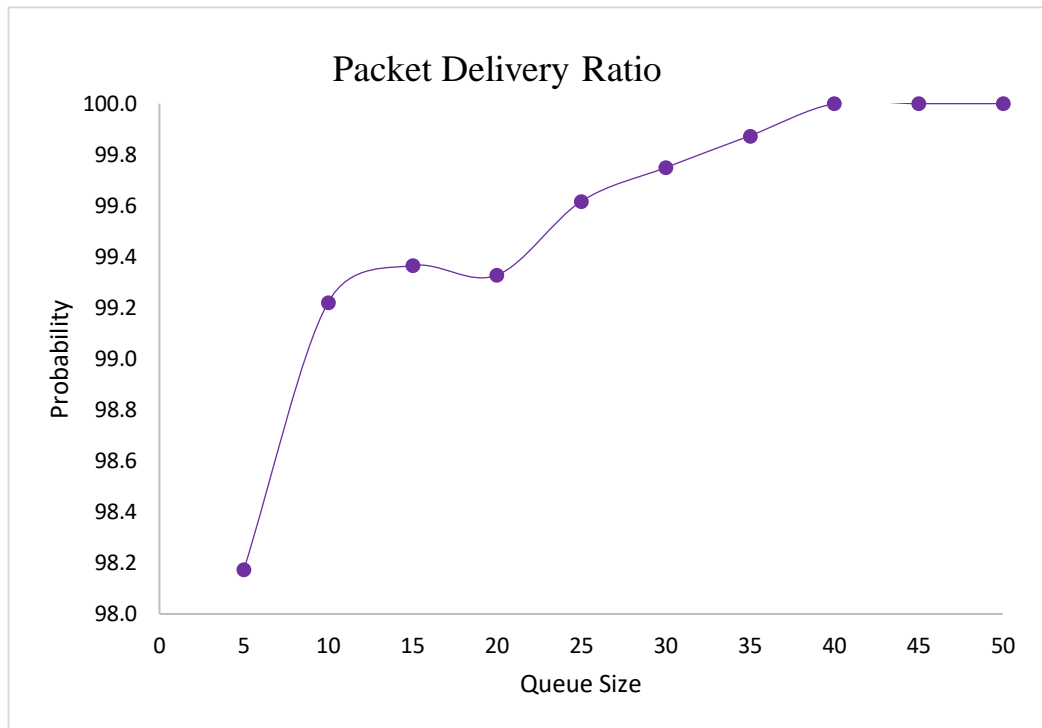


Figure 6.14: Packet Delivery Ratio v/s Queue Size

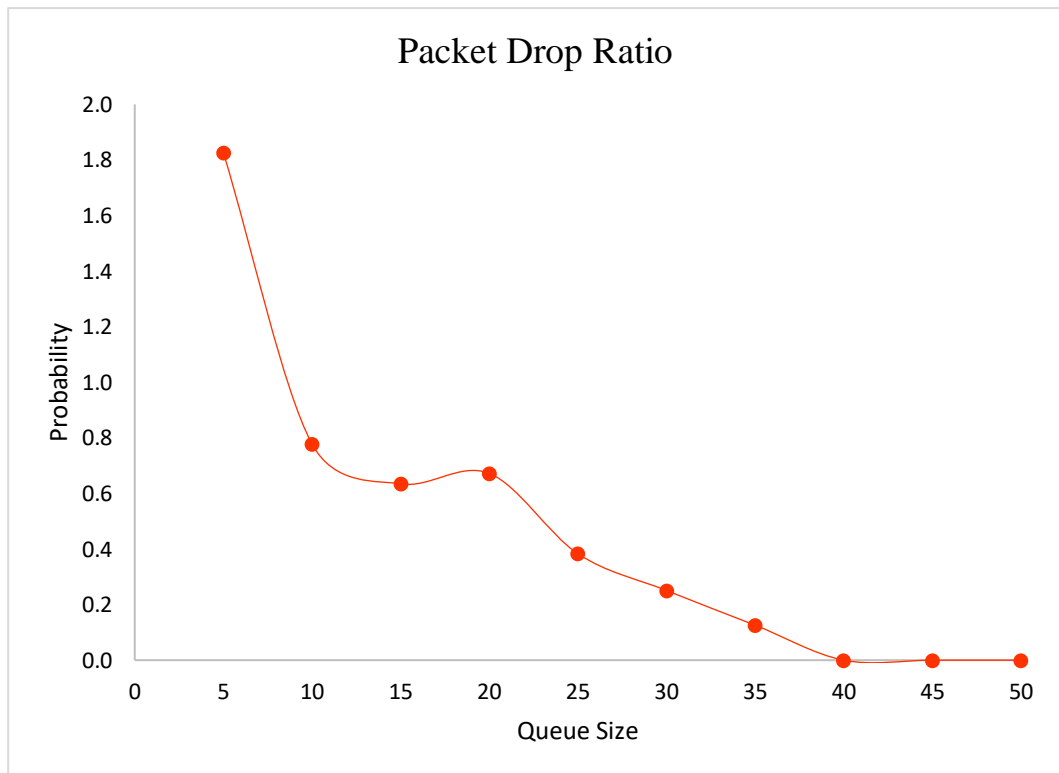


Figure 6.15: Packet Drop Ratio v/s Queue Size



### 6.5.5 Correlation Matrix

Correlation is used to express the association between the variables. The degree of association is measured by a correlation coefficient. In other words, correlation coefficients are used to measure the strength of a link. The correlation coefficient's number ranges from +1 to -1. A positive correlation exists if the value of one variable grows and the value of another variable similarly increases. A negative association exists when one variable's value rises while another variable's value falls. Zero indicates there is no association between the variables.

The analytical process is based on a matrix of correlation between the variables. Valuable insight can be obtained from this matrix. The matrix of correlation between various parameters (such as No. of Packet Received, Queue Size, Total No. of Packets, and Packet Size) are shown in Figure 6.16. The No. of Packet Received parameter is correlated to Queue Size is 0.389 (approx.), the Total No. of Packets is 0.964 (approx.), and the Packet Size is 0.078 (approx.) and similar to other variables.

Correlation between Parameters	No. of Packet Received	Queue Size	Total No. of Packets	Packet Size
No. of Packet Received	1			
Queue Size	0.38931286	1		
Total No. of Packets	0.964086941	0.3623451	1	
Packet Size	0.078563975	0.59669339	0.035202443	1

Figure 6.16: Correlation Matrix between various Parameters

### 6.5.6 Multiple Regression Model

In order to determine how two or more explanatory factors, affect the response variable, the multiple regression model is used. In general, the multiple regression model is defined as:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_k X_k \quad (6.5)$$

where  $y$  is a response variable,  $X_1 + X_2 + \dots + X_k$  are explanatory variables,  $\beta_0$  is a slope intercept coefficient and  $\beta_1 + \beta_2 + \dots + \beta_k$  are the coefficient of variables.

In Figure 6.17, where  $y$  (response variable) is *Packet Successfully Delivered*,  $X_1 + X_2 + \dots + X_k$  (explanatory variables) are Queue Size, Total No. of Packets sent, and Packet Size,  $\beta_0$  is a slope intercept coefficient and  $\beta_1 + \beta_2 + \dots + \beta_k$  are the coefficient of variables. Then derived an equation for packet successfully delivered by using equation 6.5 in the regression model as shown below:

*Packet Successfully Delivered*

$$= [(-442.855) + (14.928 * \text{Queue Size}) + (1.127 * \text{Total No. of Packets}) + (0.001 * \text{Packet Size})]$$

SUMMARY OUTPUT								
<i>Regression Statistics</i>								
Multiple R	0.965105275							
R Square	0.931428192							
Adjusted R Square	0.9295749							
Standard Error	149.9412509							
Observations	115							
<i>ANOVA</i>								
	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>			
Regression	3	33897605.15	11299201.72	502.580348	2.08188E-64			
Residual	111	2495544.036	22482.37871					
Total	114	36393149.18						
	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>	<i>Lower 95.0%</i>	<i>Upper 95.0%</i>
Intercept	-442.8550285	78.50175779	-5.641339009	1.3051E-07	-598.4115021	-287.2985549	-598.4115021	-287.2985549
Queue Size	14.92835822	7.410927703	2.01437105	0.046388615	0.243110246	29.6136062	0.243110246	29.6136062
Total No. of Packets	1.127131945	0.046728909	24.12065616	5.98362E-46	1.034535492	1.219728397	1.034535492	1.219728397
Packet Size	0.001805166	0.00619984	0.291163314	0.771469791	-0.01048023	0.014090562	-0.01048023	0.014090562

Figure 6.17: Result of Multiple Regression Model of Three Variables

It was observed that the P-value of packet size was more than 0.05 in Figure 6.17; signifying that packet size is not a statistically significant variable in the overall regression model.

In Figure 6.18, where  $y$  (response variable) is *Packet Suceesfully Delivered*,  $X_1 + X_2 + \dots + X_k$  (explanatory variables) are Queue Size and Total No. of Packets send,  $\beta_0$  is a slope intercept coefficient and  $\beta_1 + \beta_2 + \dots + \beta_k$  are coefficient of variable. Then derived an equation for packet successfully delivered from equation 6.5 as shown below:

$$\begin{aligned} & \text{Packet Suceesfully Delivered} \\ & = [(-432.250) + (16.1596 * \text{Queue Size}) \\ & \quad + (1.1207 * \text{Total No. of Packets})] \end{aligned}$$

SUMMARY OUTPUT								
<i>Regression Statistics</i>								
Multiple R	0.965078142							
R Square	0.93137582							
Adjusted R Square	0.930150389							
Standard Error	149.327361							
Observations	115							
<i>ANOVA</i>								
	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>			
Regression	2	33895699.18	16947849.59	760.0389	6.9618E-66			
Residual	112	2497450.004	22298.66075					
Total	114	36393149.18						
	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>	<i>Lower 95.0%</i>	<i>Upper 95.0%</i>
Intercept	-432.2503946	69.25658814	-6.241289186	7.96384E-09	-569.4734464	-295.0273429	-569.4734464	-295.0273429
Queue Size	16.15969764	6.060896855	2.666222182	0.008806209	4.150807438	28.16858784	4.150807438	28.16858784
Total No. of Packets	1.120753203	0.041106187	27.26482995	2.89615E-51	1.039306562	1.202199844	1.039306562	1.202199844

Figure 6.18: Result of Multiple Regression Model of Two Variables

The Multiple regression model shows 92.95% of the variation in Packet Successfully Delivered in the network can be statistically explained with Queue Size, Total No. of Packets, and Packet Size in Figure 6.17 and Figure 6.18 shows 93.015% of the variation in Packet Successfully Delivered in the network can be statistically explained with Queue Size and Total No. of Packets in multiple regression model.

### **6.5.7 Performance Evaluation of Controllers**

In proposed scheme uses the queuing technique in SDN that improves the performance of the network in round-trip time. The Round-trip Time (RTT) is a time interval between to initiated the request from starting point and receiving a response to the destination. It is measured in milliseconds (ms). The round-trip time is a vital metric for determining the condition of the network, so the network administrator to detect or monitor the speed and reliability of the network connection. The Content Delivery Network (CDN) has a primary goal to lessen the RTT of a network. If the value of RTT is reduced then the latency of the network is improved. There are several factors affecting round-trip time (RTT) value such as transmission medium, the response time of the server, the physical distance between nodes, network traffic, etc; these factors increase the congestion and slow down the network connection that hikes the value of RTT in a network.

In Software Defined Networks, the controller can be classified into two categories like single controller and multiple controllers as shown in Figure 6.19. But a single controller increases the maximum chance of failure in the network. If the controller fails for any reason, then an entire network becomes collapses or halts. To reduce or eliminate a SPOF in the network by using multiple controllers; further, has provided two options either using equal controllers or master-slave controllers in a distributed environment.

In simulation install all necessary tools such as the Ubuntu Operating System 18.04 Desktop, Mininet, and Ryu controller respectively. Now to compare the various role of SDN controllers' performance in terms of the round-trip time (RTT) with the proposed model. During simulation to analyses and evaluate the result of SDN controllers is summarized in Table 20 and shown in Figures (6.20 to 6.23).

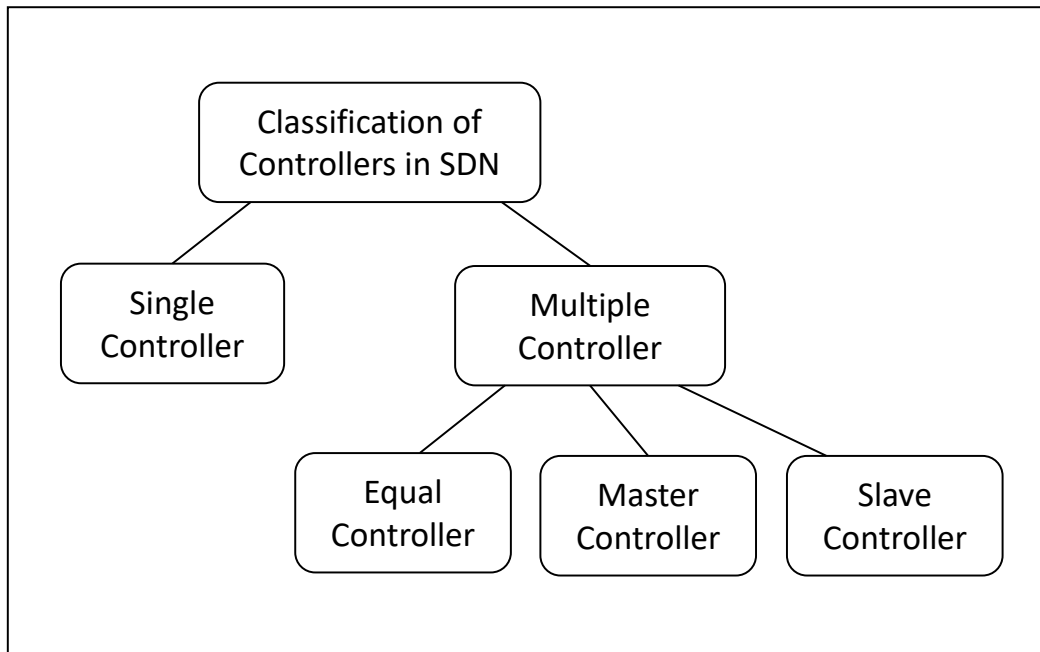


Figure 6.19: Classification of Controllers in Software Defined Network

Table 20: Performance Comparison of Controllers in Round-trip Time Metric

<b><i>Round-Trip Time (RTT)</i></b>	<b><i>Min (ms)</i></b>	<b><i>Avg (ms)</i></b>	<b><i>Max (ms)</i></b>	<b><i>Mdev (ms)</i></b>
<b><i>Single Controller</i></b>	0.042	43.522	2062.932	268.729
<b><i>Equal Controller</i></b>	0.081	41.889	94.402	32.708
<b><i>Master-Slave Controller</i></b>	0.073	0.473	36.974	3.670
<b><i>Proposed Model</i></b>	0.074	0.100	0.450	0.039

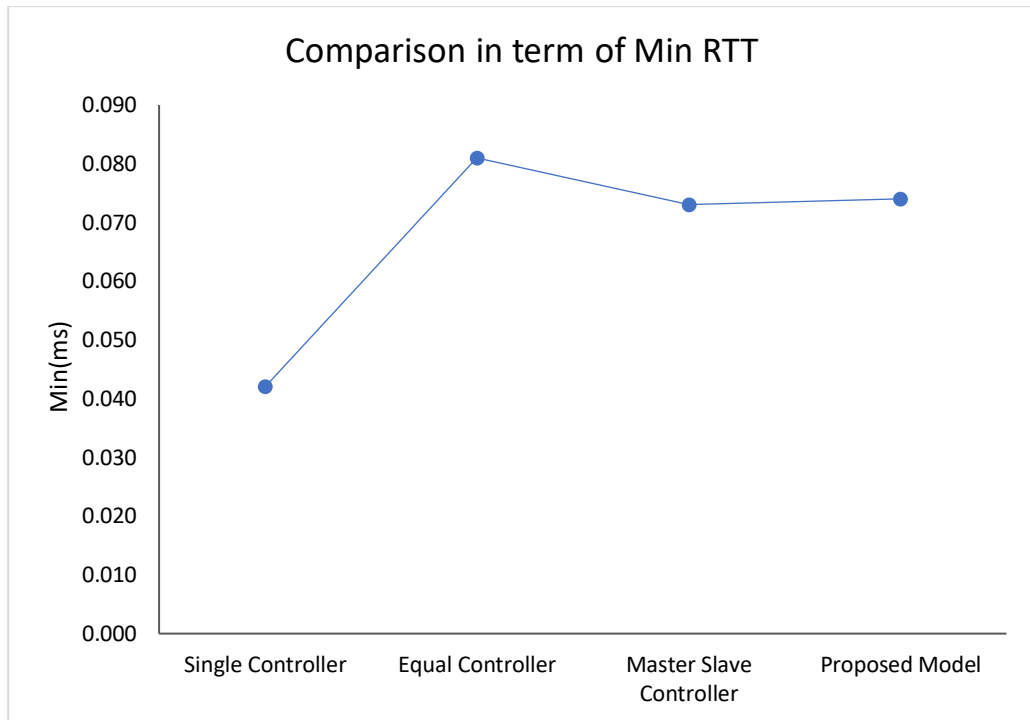


Figure 6.20: Comparison between Controllers in term Minimum RTT

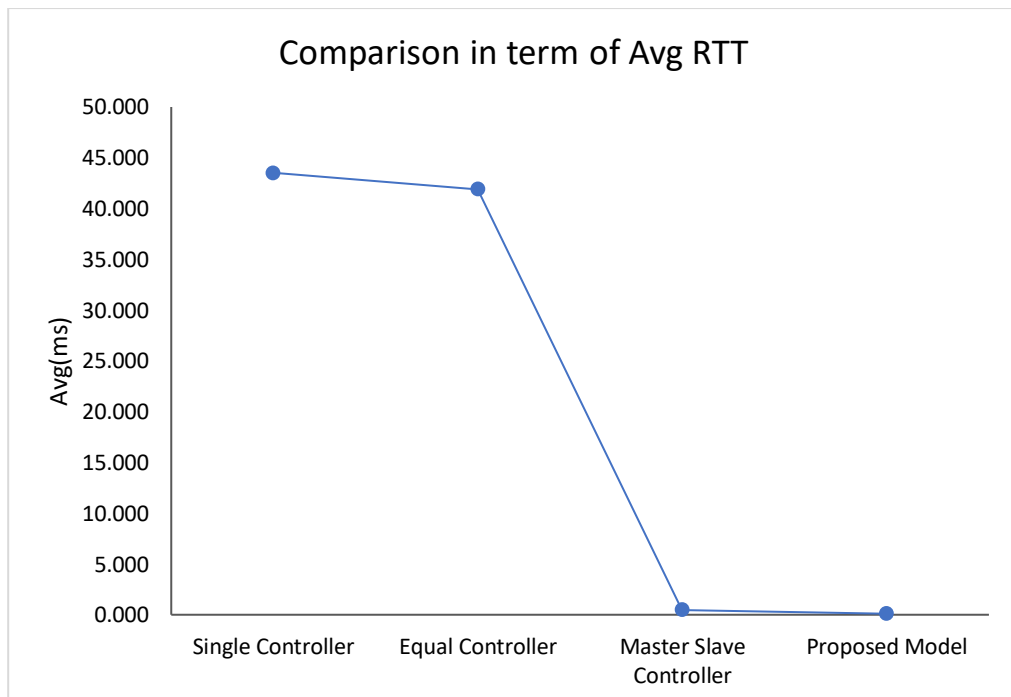


Figure 6.21: Comparison between Controllers in term Average RTT

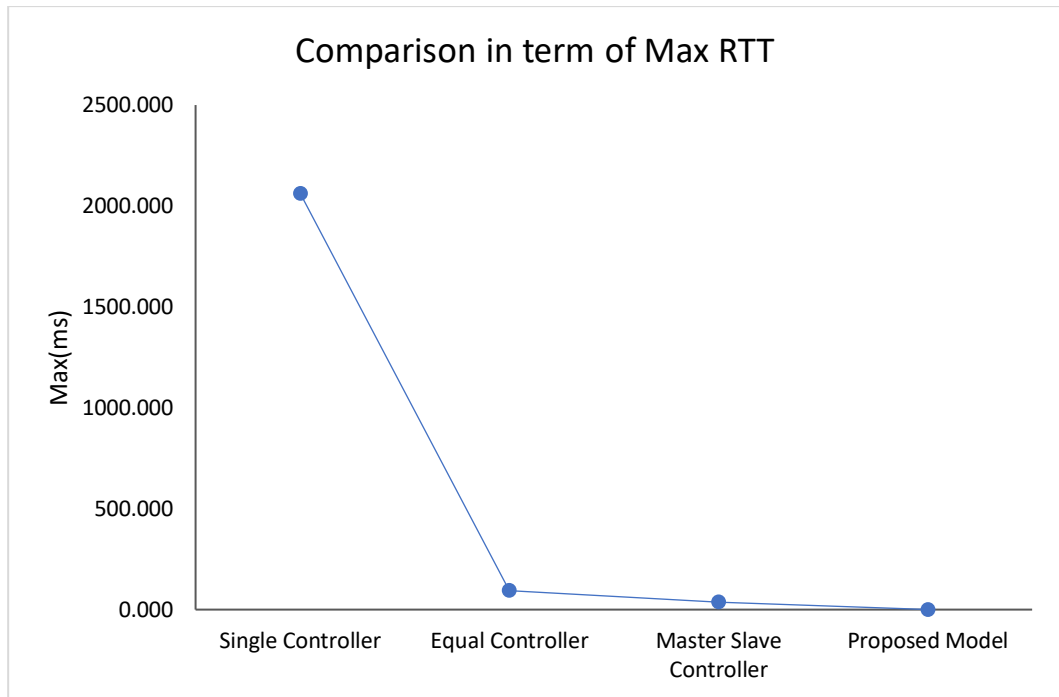


Figure 6.22: Comparison between Controllers in term Maximum RTT

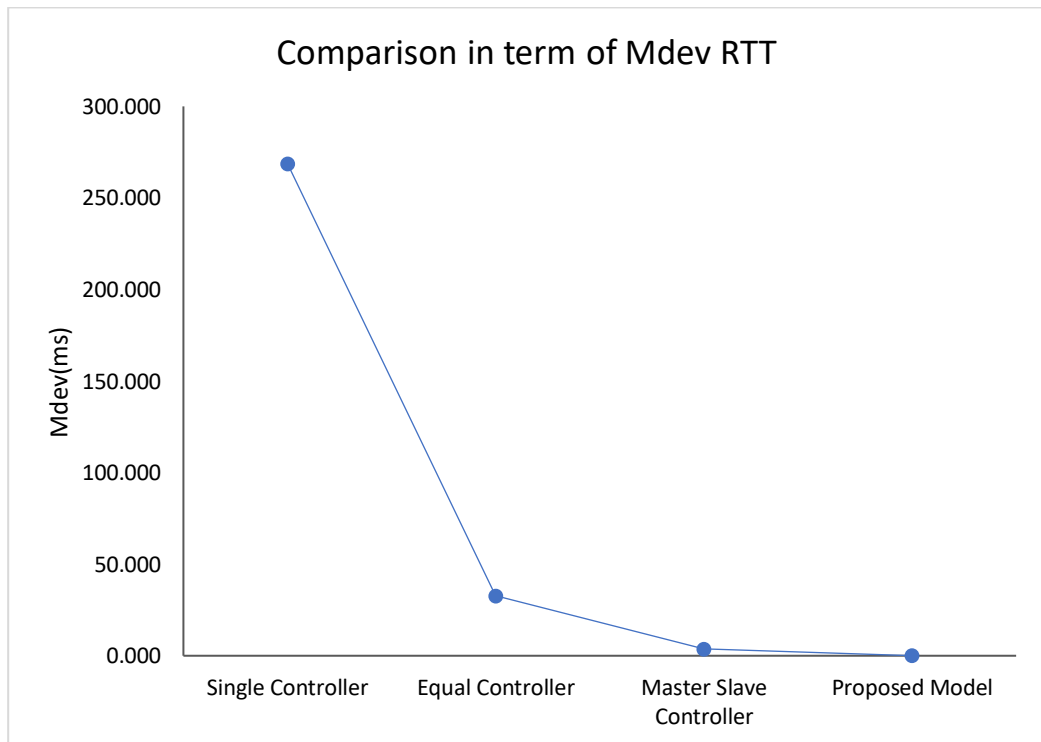


Figure 6.23: Comparison between Controllers in term Mean Deviation RTT

A performance metric is a function of a parameter that quantifies its influence; that parameter affects the system metrics, in other words. Applying or measuring performance analysis is required to ascertain how a performance metric relates to a system. Analysing the computing system's performance is what it entails. It involves looking at how well the computer system performs. To evaluate the SDN controllers' performance using the D-ITG traffic generator in conjunction with the suggested model for operation. In Table 21 and graphically in Figures 6.24 to 6.27, the whole list of evaluation criteria is displayed.

Table 21: Performance Evaluation of Controllers with the Proposed Model

Operation of SDN Controller	Evaluation of Parameters								
	Min Delay	Max Delay	Average Delay	Average Jitter	Delay Standard Deviation	Bytes Received	Average Bitrate	Average Packet Rate	Time Duration (Sec)
Single Controller	0.000047	0.000443	0.000156	0.000011	0.000051	985500	394.2900	98.5720	20
Equal Controller	0.000029	0.000446	0.000192	0.000010	0.000052	982500	393.0110	98.2529	20
Master Slave Controller	0.000029	0.000376	0.000154	0.000008	0.000051	985500	394.2445	98.5611	20
Proposed Model	0.000024	0.000423	0.000118	0.000010	0.000031	987500	395.0126	98.7531	20



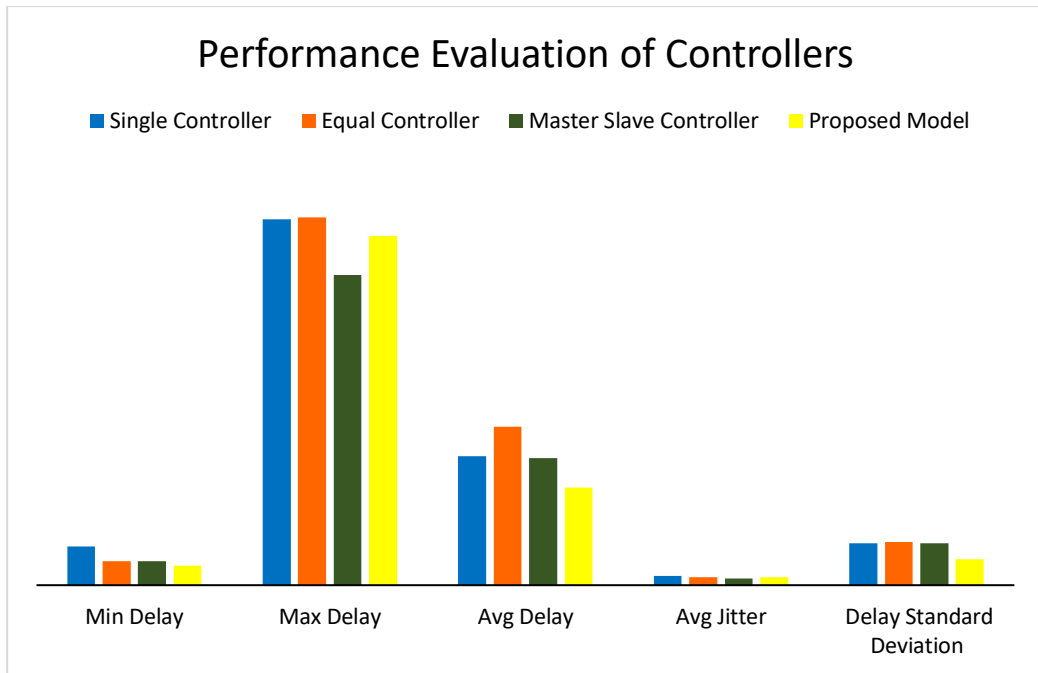


Figure 6.24: Performance Evaluation w.r.t. Min Delay, Max Delay, Avg Delay, Avg Jitter and Delay Standard Deviation

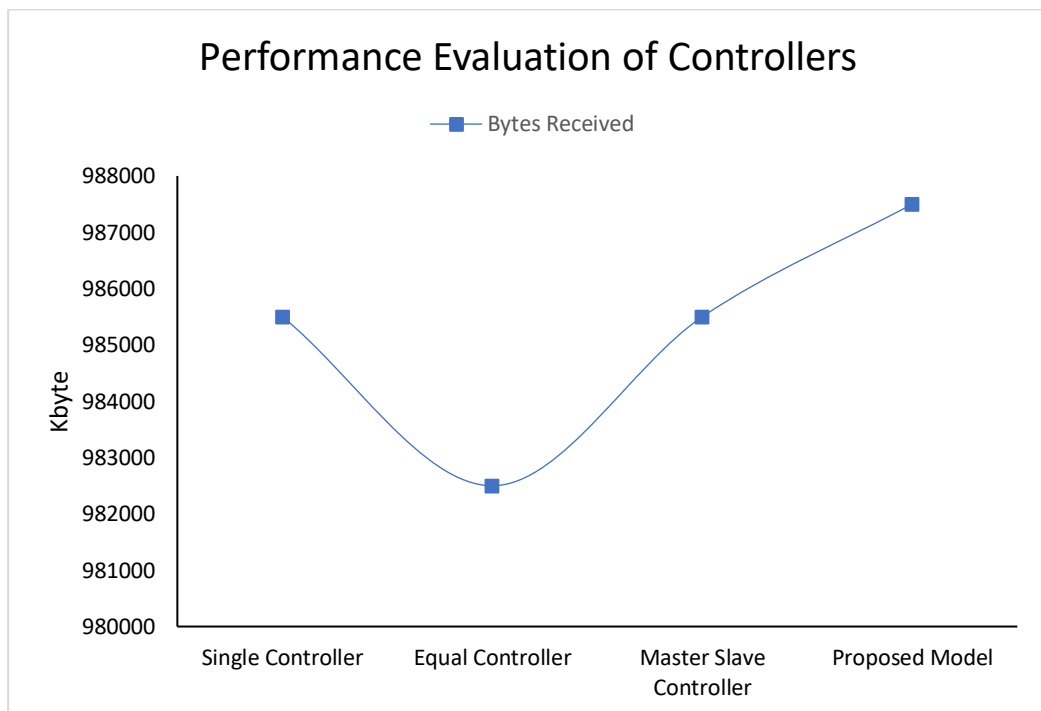


Figure 6.25: Performance Evaluation w.r.t. Bytes Received

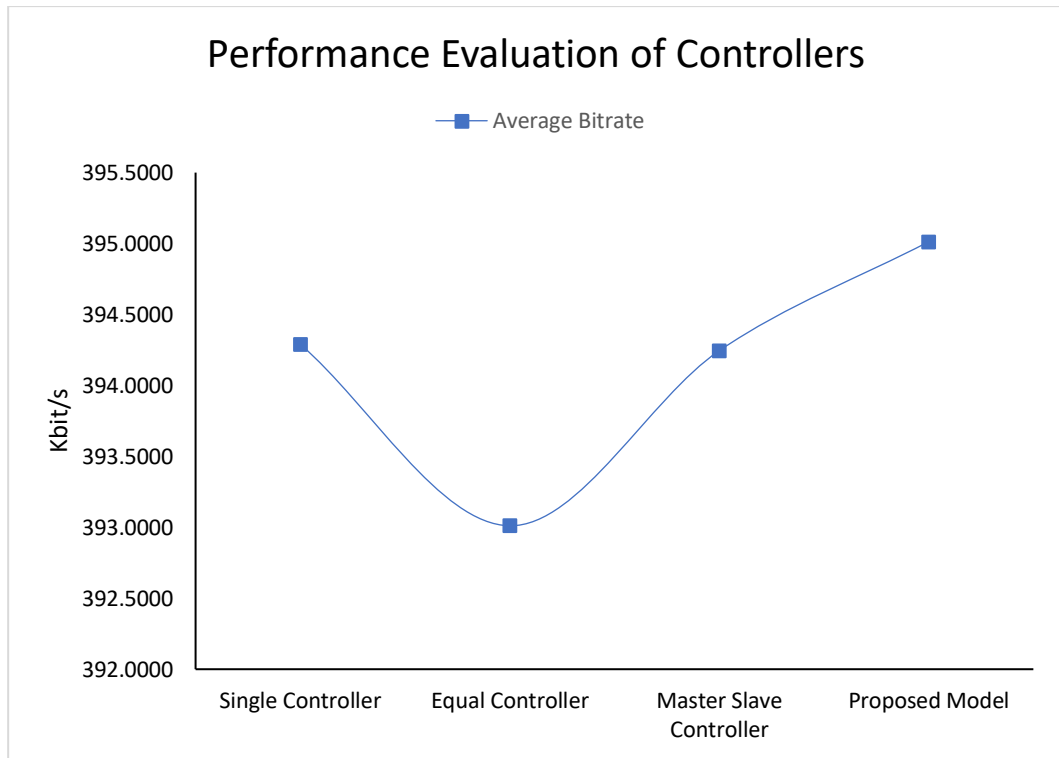


Figure 6.26: Performance Evaluation w.r.t. Average Bitrate

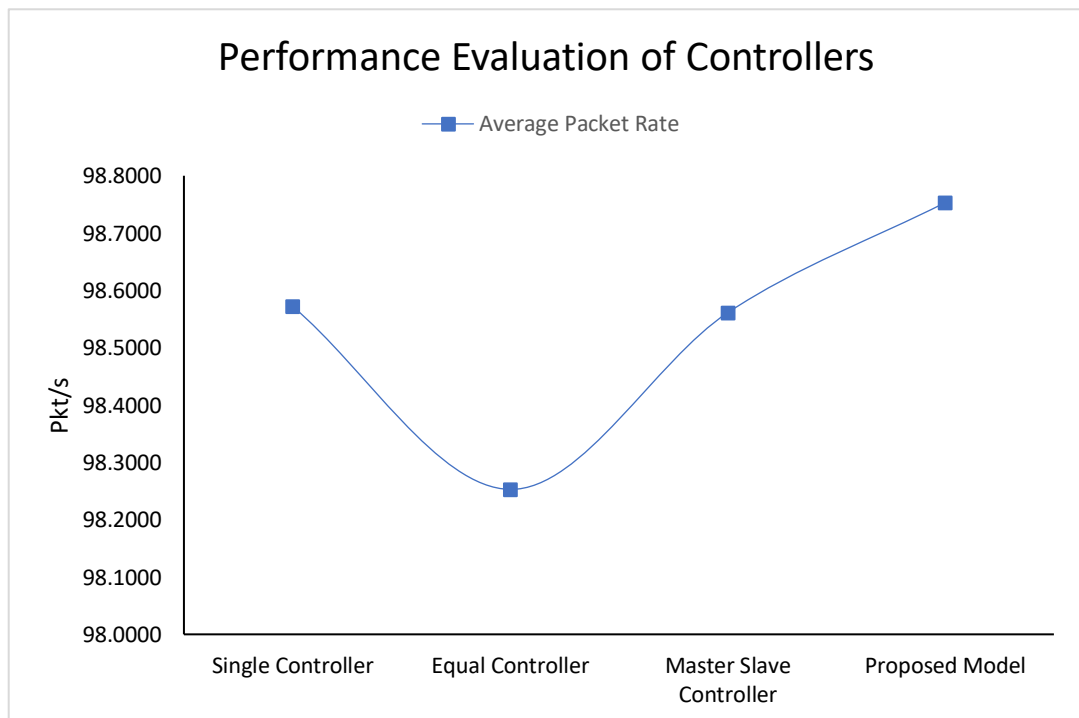


Figure 6.27: Performance Evaluation w.r.t. Average Packet Rate

When compared to the other types of controllers in Table 20, the proposed model's minimum, average, and delay standard deviation parameters have the lowest values. But the parameters for bytes received, bitrate, and average packet rate are higher than in other controller configurations.

To assess the effectiveness of SDN controllers in their roles with the proposed model, the iperf tool is used. As shown in Figures 6.28 to 6.31 and Table 22, compare controllers based on various metrics, including average throughput, average bandwidth, and ping delay. Table 22 depicts the operation of controllers with the proposed model. In comparison to alternative controller configurations, the proposed model has higher average throughput and bandwidth metrics and lower ping delay metrics.

Table 22: Performance Evaluation of Controllers with the Proposed Model w.r.t. Average Throughput, Average Bandwidth and Ping Delay

<b>Operation of SDN Controller's</b>	<b>Average Throughput (GBytes)</b>	<b>Average Bandwidth (Gbits/sec)</b>	<b>Ping Delay (ms)</b>	<b>Time Interval (Sec)</b>
Single Controller	63.5	36.4	43.480	15
Equal Controller	69.8	40.0	41.808	15
Master Slave Controller	70.9	40.6	0.400	15
Proposed Model	71.4	40.9	0.026	15

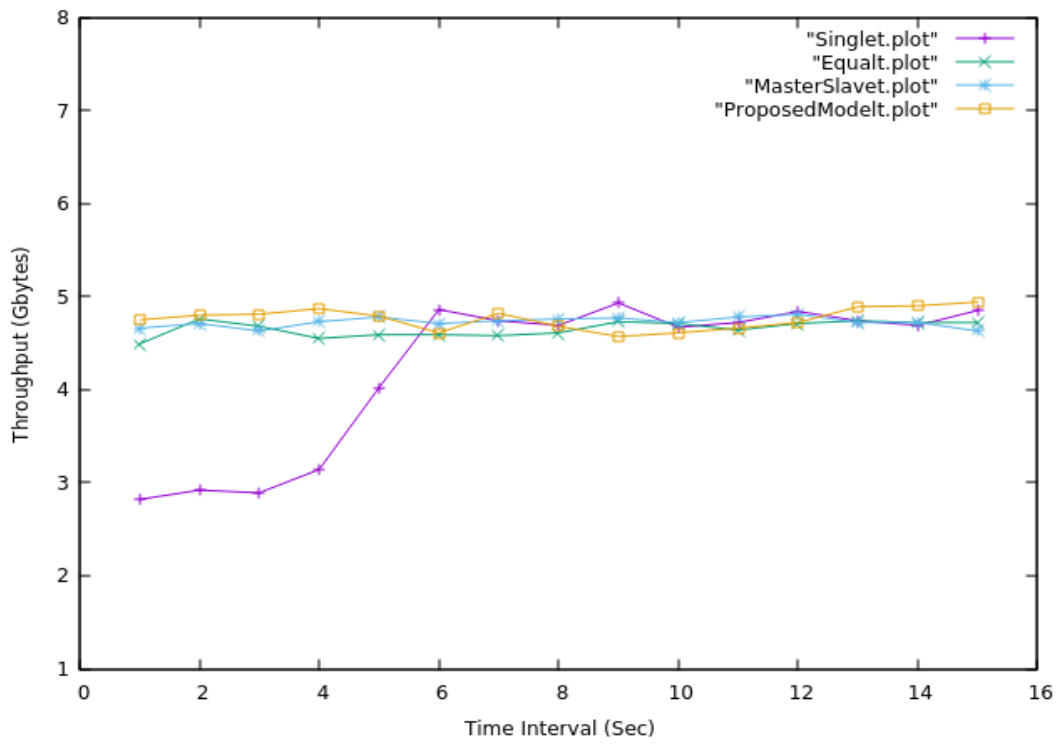


Figure 6.28: Performance Evaluation of Controllers w.r.t. Throughput by Gnuplot

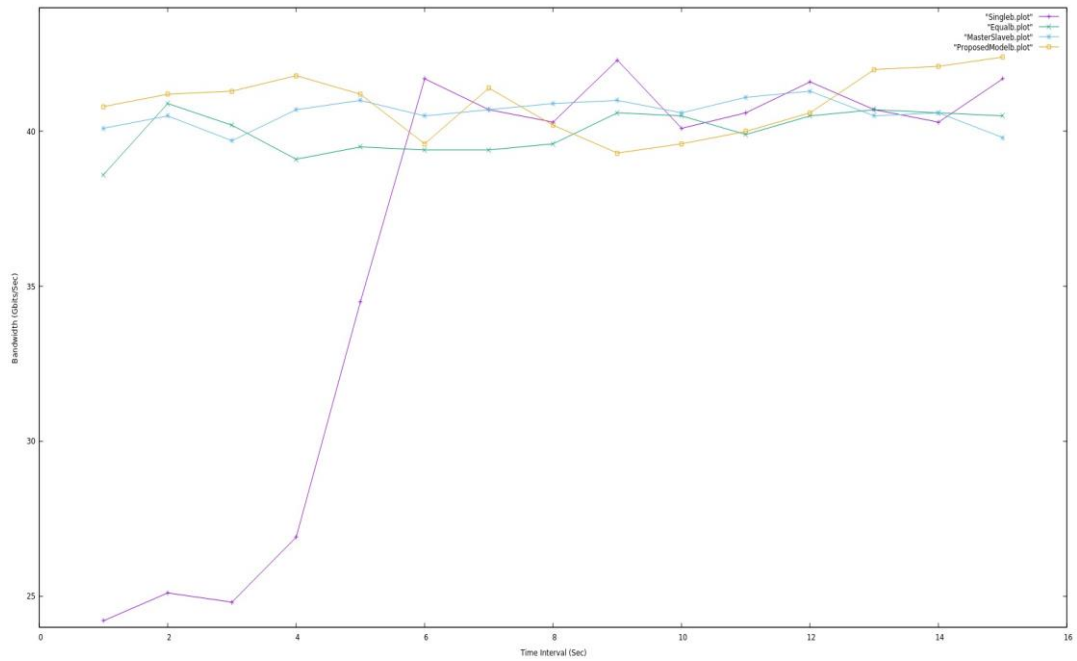


Figure 6.29: Performance Evaluation of Controllers w.r.t. Bandwidth by Gnuplot

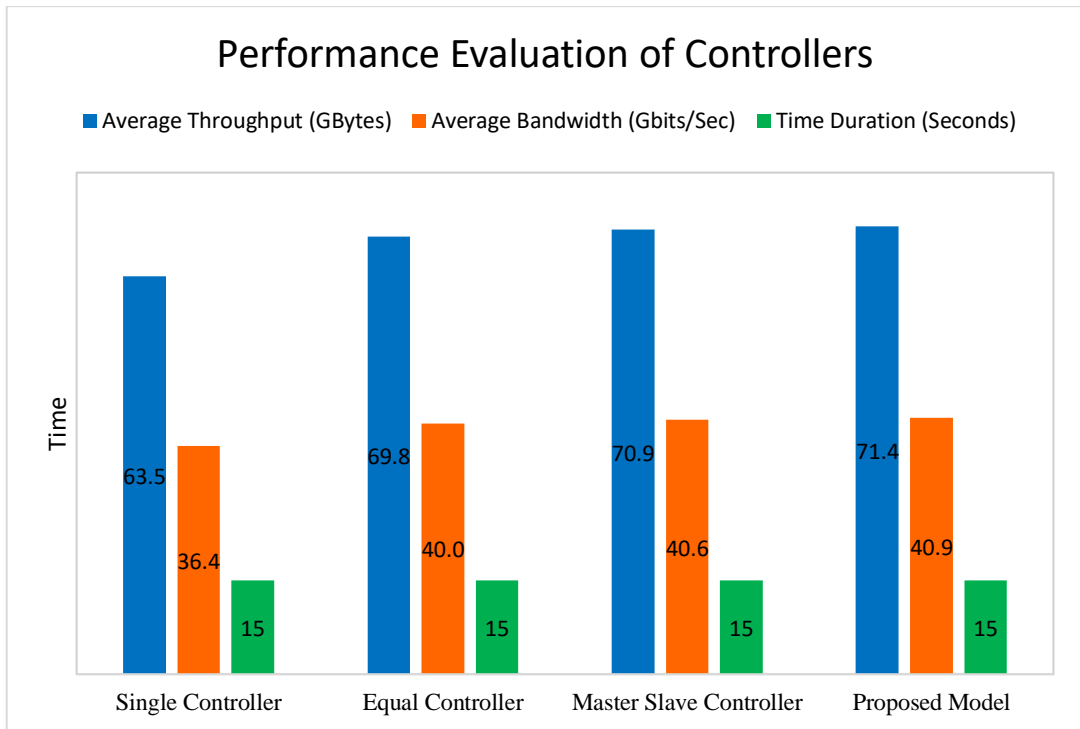


Figure 6.30: Performance Evaluation of Controllers with the Proposed Model

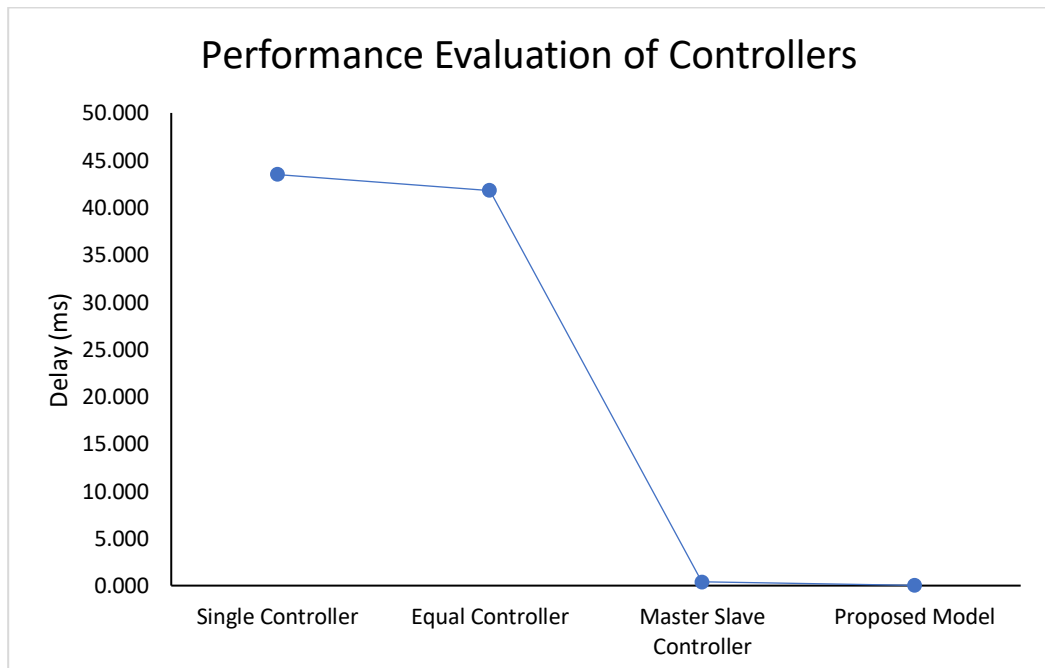


Figure 6.31: Performance Evaluation of Controllers w.r.t. Ping Delay

Table 23 provides a quantitative analysis of the proposed model's parameters in terms of the percentage of improvement over other types of controllers. After looking at Tables 21, 22, and 23, it's clear that the performance of the proposed model is better and superior to that of other SDN controller configurations.

Table 23: Quantitative Analysis of Parameters

Percentage of Improvement w.r.t. Proposed Model →	Parameters		
	<i>Throughput</i>	<i>Bandwidth</i>	<i>Packet Rate</i>
Single Controller	12.44%	12.36%	0.184%
Equal Controller	2.29%	2.25%	0.509%
Master Slave Controller	0.71%	0.74%	0.195%

## 6.6 Conclusion

The Software Defined Network provides a novel paradigm of networking that enhances innovation in networking as compared to the traditional network. The main contribution is to propose an adaptive algorithm for load balancing in multiple controllers by using the queuing technique with the Markov chain to evaluate an equilibrium/steady state of a probability distribution for controllers, which assists manage the load among controllers in a convenient way. Based on probability, it reduces packet dropping or packet lost ratio, overheads, and migration cost of the network due to managing load balancing. As a consequence, a cascading failure of controllers in a network that occurs due to an imbalance of controllers can be avoided. That implies the multiple controllers provide a ubiquitous and robust network that extends the scalability, dependability, and high availability of a network service after evaluating the equilibrium state of a probability distribution of

controllers. In comparison to the other types of controllers in Table 21, the proposed model's average delay and delay standard deviation parameters have the lowest values. Table 23 is a quantitative analysis of the proposed model's parameters in terms of how much better they are than other kinds of controllers. Moreover, the proposed model's performance is more appropriate as compared to other controllers.

# **Chapter 7**

## **Conclusion and Future Work**



# 7 CHAPTER

## Conclusion and Future Work

### 7.1 Conclusion

The dissociation of the two planes is the primary factor in the Software Defined Network's ability to offer a novel and agile networking paradigm that fosters innovation in networking compared to the traditional network. A communication system's important feature today is the availability of services on-demand and without interruption. The firm will suffer a significant loss of revenue or profit if there is any disruption or failure in the communication system. By expanding the availability of networking facilities, this problem must be solved.

Designing a fault tolerance model that decreases the possibility of failure by one point in an SDN network is the aim of this research. When an SDN controller fails for whatever cause, it is analysed through simulation that the entire network becomes unreachable; this means that the network is ruined until the controller is available in the network and in a functioning state. As a result, a single controller has a greater effect on network failure. For any type of communication system, it is not feasible. To solve this issue, a fault tolerance mechanism that uses multiple controllers to reduce the likelihood that the network will have a single or singular point of failure is needed.

The most difficult task in a distributed environment is managing load balancing between controllers. Moreover, both fault tolerance and load balancing are complicated and interrelated issues when dealing with the multiple controllers in

SDN. To resolve these issues, using the Queuing Theory Technique and Markov Continuous Chain helps manage the fluctuation of load between the multiple controllers. A suggested adaptive for load balancing in multiple controllers, which uses the queuing technique with the Markov chain to evaluate an equilibrium or steady state of a probability distribution for controllers and aids in managing the load among controllers, is useful for achieving this goal. These probabilities provide an idea of how to distribute network traffic among controllers, and managing load balancing reduces packet dropping or packet loss ratio, overheads, and network migration cost. As a result, a cascading failure of controllers in a network caused by an imbalance in controller workload can be avoided. Based on probability, it also reduces overheads that are induced in the switch migration process and the migration cost of the network and minimises the maintenance cost of the network.

In order to statistically explain the variance in Packet Successfully Delivery in a network with respect to the Queue Size, Total No. of Packets, and Packet Size parameters in the network, evaluate the correlation matrix and the multiple regression model. After assessing the equilibrium state of a probability distribution of controllers, it is implied that the numerous controllers offer a widespread and reliable network that increases the scalability, dependability, and high availability of a network service. The proposed model then demonstrated how superior it was in comparison to other types of controllers. The next part of the chapter talks about how the research presented in this thesis can be used to do more work.

## **7.2 Future Work**

The Software Defined Network can resolve the problems with the present or traditional networking system, despite having a unique networking architecture. SDN can segregate the data plane from the control plane; therefore, overall control of the network is transferred to the SDN controller. A real world is a vast and extremely complex system. Because it offers the capability of demonstrating the virtual network in order to understand how it functions in the real world, many large organisations are converting or transforming their systems to Software Defined

Networks. The goal of this thesis is to manage load balancing between multiple controllers in an SDN network based on the equilibrium state of the distribution between the controllers, as well as to improve fault tolerance in the control plane by eliminating a single point of failure through the use of multiple controllers.

In terms of design and analysis, managing a network with several controllers is far more difficult and complex than managing a network with a single controller. A large number of simulations can be performed using the work that has been proposed in order to do performance evaluations and a quantitative analysis of the performance parameters of the proposed model in terms of how much better they are than other types of controllers.

The additional work that can be done using the research presented in this thesis is briefly described in the part that follows. In the future, make an effort to implement the right security measures for the controller, as it is tasked with managing the entire network. A good security mechanism helps to prevent attempts by hackers to take over the controller on the control plane. Additionally, it is thought that the synchronisation between the controllers will manage any inconsistencies. In order to increase the availability of the networks, attempt to keep data synchronised between the controllers. The effort described in the thesis, they believe, will improve the standard of service provided by the networking system.

# References

## References

- [1] S. Hanji et al., “Comparison of Software Defined Networking with Traditional Networking,” *Asian Journal of Research in Computer Science*, vol. 9, no. 2, pp. 1-18, 2021.
- [2] D. Gopi, S. Cheng and R. Huck, “Comparative Analysis of SDN and Conventional Networks using Routing Protocols,” in *2017 International Conference on Computer, Information and Telecommunication Systems (CITS)*, 2017.
- [3] M. J. Anjum, I. Raza and S. A. Hussain, “Real-Time Multipath Transmission Protocol (RMTP): A Software Defined Networks (SDN) based Routing Protocol for Data Centric Networks,” in *1<sup>st</sup> International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, pp. 1-6, 2019.
- [4] M. Y. Daha, M. S. M Zahid, K. Husain and F. Ousta, “Performance Evaluation of Software Defined Networks with Single and Multiple Link Failure Scenario under Floodlight Controller,” in *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pp. 959-965, 2021.
- [5] G. Khetrapal and S. K. Sharma, “Demystifying Routing Services in Software-Defined Networking,” *Aricent Demystifying Routing Services SDN*, pp. 1-12, 2014.
- [6] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
- [7] H. Zhang and J. Yan, “Performance of SDN Routing in Comparison with Legacy Routing Protocols,” in *2015 International Conference on Cyber-*

- Enabled Distributed Computing and Knowledge Discovery, pp. 491-494, 2015.
- [8] W. Braun and M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Open Access Future Internet*, vol. 6, no. 2, pp. 302-336, 2014.
- [9] S. Khan, A. Wahid and S. Tanvir, "Comparative Study of Routing Strategies in Software Defined Networking," in *SAC '16: Proceedings of the 31<sup>st</sup> Annual ACM Symposium on Applied Computing*, pp. 696-702, 2016.
- [10] I. Radu, P. Ciotirnae and F. Popescu, "Integrating Software Defined Networks with Traditional Hardware Networks," in *2018 International Conference on Communications (COMM)*, pp. 309-312, 2018.
- [11] Z. Hu, M. Wang, X. Yan, Y. Yin and Z. Luo, "A Comprehensive Security Architecture for SDN," in *2015 18<sup>th</sup> International Conference on Intelligence in Next Generation Networks*, 2015.
- [12] I. Bedhief, M. Kassar and T. Aguilu, "SDN-based Architecture Challenging the IoT Heterogeneity," in *2016 3<sup>rd</sup> Smart Cloud Networks & Systems (SCNS)*, 2016.
- [13] C. Tselios, I. Politis and S. Kotsopoulos, "Enhancing SDN Security for IoT-related deployments through Blockchain," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2017.
- [14] L. Shif, F. Wang and C. Lung, "Improvement of Security and Scalability for IoT Network Using SD-VPN," in *NOMS 2018 – 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018.
- [15] Y. Yu, X. Li, X. Leng, L. Song, K. Bu, Y. Chen, J. Yang, L. Zhang, K. Cheng and X. Xiao, "Fault Management in Software-Defined Networking: A

- Survey,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 349-392, 2018.
- [16] A. Malik, B. Aziz, A. Al-Haj and M. Adda, “Software-Defined Networks: A Walkthrough Guide From Occurrence To Data Plane Fault Tolerance,” *PeerJ Preprints Open Access*, pp. 1-26, 2019.
- [17] M. R. Parsaei, S. H. Khalilian and R. Javidan, “A Comparative Study on Fault Tolerance Methods in IP Networks versus Software Defined Networks,” *International Academic Journal of Science and Engineering*, vol. 3, no. 4, pp. 146-154, 2016.
- [18] J. Chen, J. Chen, F. Xu, M. Yin and W. Zhang, “When Software Defined Networks Meet Fault Tolerance: A Survey,” in G. Wang et al. (Eds): *International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2015)*, Part III, vol. 9530, pp. 351-368, 2015.
- [19] C. M. Duran, E. A. Leal and J. F. Botero, “Improving fault tolerance in critical networks through OpenFlow,” in *2017 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2017.
- [20] W. Li, W. Meng and L. F. Kwok, “A Survey on OpenFlow-based Software Defined Networks: Security Challenges and Countermeasures,” *Journal of Network and Computer Applications*, vol. 68, pp. 126-139, 2016.
- [21] B. Isong, I. Mathebula and N. Dladlu, “SDN-SDWSN Controller Fault Tolerance Framework for Small to Medium Sized Networks,” in *2018 19<sup>th</sup> IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, IEEE Computer Society, pp. 43-51, 2018.
- [22] S. Vissicchio, L. Vanbever and O. Bonaventure, “Opportunities and Research Challenges of Hybrid Software Defined Networks,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 70-75, 2014.

- [23] T. Chen, M. Matinmikko, X. Chen, X. Zhou and P. Ahokangas, "Software Defined Mobile Networks: Concept, Survey, and Research Directions," *IEEE Communications Magazine*, vol. 53, no. 11, pp. 126-133, 2015.
- [24] I. F. Akyildiz, A. Lee, P. Wang, M. Luo and W. Chou, "Research Challenges for Traffic Engineering in Software Defined Networks," *IEEE Network*, vol. 30, no. 3, pp. 52-58, 2016.
- [25] M. Casado, T. Garfinkel, A. Akella, M. Freedman, D. Boneh and N. McKeown, "SANE: A Protection Architecture for Enterprise Networks," in *Proceedings of the 15<sup>th</sup> Conference on USENIX Security Symposium*, vol. 15, pp. 137-151, 2006.
- [26] O. Akonjang, "SANE: A Protection Architecture for Enterprise Networks," pp. 1-10, 2007.
- [27] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, 2008.
- [28] M. Paliwal, D. Shrimankar and O. Tembhurne, "Controllers in SDN: A Review Report," *IEEE Access*, vol. 6, pp. 36256-36270, 2018.
- [29] A. Nantoume, B. Kone, A. D. Kora and B. Niang, "Software Defined Networking (SDN) for Universal Access," in *International Conference on Emerging Technologies for Developing Countries (AFRICATEK 2018)*, pp. 133-144, 2018.
- [30] G. Araniti, J. Cosmas, A. Lera, A. Molinaro, R. Morabito and A. Orsino, "OpenFlow over Wireless Networks: Performance Analysis," in *2014 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, 2014.



- [31] A. Kondel and A. Ganpati, "Evaluating System Performance for handling scalability challenge in SDN," in 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), 2015.
- [32] A. Ghosh and T. Manoranjitham, "A study on load balancing techniques in SDN," *International Journal of Engineering & Technology*, vol. 7, no. 2.4, pp. 174-177, 2018.
- [33] S. Scott-Hayward, S. Natarajan and S. Sezer, "A Survey of Security in Software Defined Networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623-654, 2016.
- [34] M. Casado, N. Foster and A. Guha, "Abstractions for Software-Defined Networks," *Communications of the ACM*, vol. 57, no. 10, pp. 86-95, 2014.
- [35] S. Narayana, J. Rexford and D. Walker, "Compiling Path Queries in Software-Defined Networks," in *HotSDN '14*, 2014.
- [36] R. Wang, D. Butnariu and J. Rexford, "OpenFlow-Based Server Load Balancing Gone Wild," in 11<sup>th</sup> USENIX Conference on Hot topics in management of internet, cloud, and enterprise networks and services, pp. 1-12, 2011.
- [37] X. Xu and L. Hu, "A Software Defined Security Scheme Based on SDN Environment," in 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), pp. 504-512, 2017.
- [38] W. Liao, S. Kuai and C. Lu, "Dynamic Load-Balancing Mechanism for Software-Defined Networking," in 2016 International Conference on Networking and Network Applications (NaNA), pp. 336-341, 2016.
- [39] A. S. Nugroho, Y. D. Safitri and T. A. Setyawan, "Comparison Analysis of Software Defined Network and OSPF Protocol Using Virtual Media," in

- 2017 IEEE International Conference on Communication, Networks and Satellite (Comnetsat), pp. 106-111, 2017.
- [40] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi and M. Conti, "A Survey on the Security of Stateful SDN Data Planes," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1701-1725, 2017.
- [41] S. Bera, S. Misra and M. Obaidat, "Mobility-Aware Flow-Table Implementation in Software-Defined IoT," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016.
- [42] J. Jiang, H. Huang, J. Liao and S. Chen, "Extending Dijkstra's Shortest Path Algorithm for Software Defined Networking," in *16<sup>th</sup> Asia-Pacific Network Operations and Management Symposium*, 2014.
- [43] L. Kra, Y. Gondo, B. T. Goore and O. Asseu, "Contribution to the Optimization of the Energy Consumption in SDN Networks," *Journal of Sensor Technology*, vol. 8, no. 3, pp. 59-67, 2018.
- [44] T. Luo, H. Tan and T. S. Quek, "Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks," *IEEE Communications Letters*, vol. 16, no. 11, pp. 1896-1899, 2012.
- [45] T. Theodorou and L. Mamatas, "CORAL-SDN: A Software-Defined Networking Solution for the Internet of Things," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2017.
- [46] K. Wang, S. Kao and M. Kao, "An Efficient Load Adjustment for Balancing Multiple Controllers in Reliable SDN Systems," in *2018 IEEE International Conference on Applied System Invention 2018 (ICASI)*, pp. 593-596, 2018.
- [47] D. Wu, D. Arkhipov, E. Asmare, Z. Qin and J. McCann, "UbiFlow: Mobility Management in Urban-scale Software Defined IoT," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015.

- [48] M. C. Nkosi, A. A. Lysko and S. Dlamini, "Multi-path Load Balancing for SDN Data Plane," in 2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC), 2018.
- [49] I. P. A. Suwandika, M. A. Nugroho and M. Abdurahman, "Increasing SDN Network Performance using Load Balancing Scheme on Web Server," in 2018 6<sup>th</sup> International Conference on Information and Communication Technology (ICoICT), 2018.
- [50] H. Kim, J. R. Santos, Y. Turner, M. Schlansker, J. Tourrilhes and N. Feamster, "CORONET: Fault Tolerance for Software Defined Networks," 2012 20<sup>th</sup> IEEE International Conference on Network Protocols (ICNP), 2012.
- [51] P. C. Fonseca and E. S. Mota, "A Survey on Fault Management in Software-Defined Networks," IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2284-2321, 2017.
- [52] F. Bannour, S. Souihi and A. Mellouk, "Distributed SDN Control: Survey, Taxonomy and Challenges," IEEE Communications Surveys & Tutorials, vol. 20, no. 1, pp. 333-354, 2017.
- [53] S. Bera, S. Misra, S. K. Roy and M. S. Obaidat, "Soft-WSN: Software-Defined WSN Management Systems for IoT Applications," IEEE Systems Journal, vol. 12, no. 3, pp. 2074-2081, 2018.
- [54] M. Abdullah, N. Al-awad and F. Hussein, "Performance Evaluation and Comparison of Software Defined Networks Controllers," International Journal of Scientific Engineering and Science, vol. 2, no. 11, pp. 45-50. 2018.
- [55] G. Wang, Y. Zhao, J. Huang and W. Wang, "The Controller Placement in Software Defined Networking: A Survey," IEEE Network, vol. 31, no. 5, pp. 21-27, 2017.

- [56] Y. Jimenez, C. Cervello-Pastor and A. Garcia, "On the controller placement for designing a distributed SDN control layer," in 2014 IFIP Networking Conference, 2014.
- [57] K. Kuroki, N. Matsumoto and M. Hayashi, "Scalable OpenFlow Controller Redundancy Tackling Local and Global Recoveries," in 5<sup>th</sup> International Conference on Advances in Future Internet, pp. 61-66, 2013.
- [58] A. Gonzalez, G. Nencioni, B. E. Helvik and A. Kamisinski, "A Fault-Tolerant and Consistent SDN Controller," in 2016 IEEE Global Communications Conference (GLOBECOM), 2016.
- [59] L. Sidki, Y. Ben-Shimol and A. Sadovski, "Fault Tolerant Mechanisms for SDN Controllers," in IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 1-6, 2016.
- [60] N. Medhi and D. K. Saikia, "OpenFlow-Based Multi-controller Model for Fault-Tolerant and Reliable Control Plane," in Smart Computing Paradigms: New Progresses and Challenges Proceeding of ICACNI 2018, vol. 2, pp. 61-73, 2018.
- [61] Y. E. Oktian, S. Lee, H. Lee and J. Lam, "Distributed SDN controller system: A survey on design choice," *Computer Networks*, vol. 121, pp. 100-111, 2017.
- [62] N. Katta, H. Zhang, M. Freedman and J. Rexford, "Ravana: Controller Fault-Tolerance in Software-Defined Networking", in 1<sup>st</sup> ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR' 15), pp. 1-12, 2015.
- [63] A. Mantas and F. V. Ramos, "Rama: Controller Fault Tolerance in Software-Defined Networking Made Practical," *ArXiv*, vol. abs/1902.01669, 2019.
- [64] M. Karakus and A. Durrezi, "A Survey: Control Plane Scalability Issues and Approaches in Software-Defined Networking (SDN)," *Computer Networks*, vol. 112, pp. 279-293, 2017.

- [65] O. Blial, M. B. Mamoun and R. Benaini, "An Overview on SDN Architectures with Multiple Controllers," *Journal of Computer Networks and Communications*, vol. 2016, pp. 1-8, 2016.
- [66] Y. Zhang, L. Cui, W. Wang and Y. Zhang, "A Survey on Software Defined Networking with Multiple Controllers," *Journal of Network and Computer Applications*, vol. 103, pp. 101-118, 2018.
- [67] A. U. Rehman, R. L. Aguiar and J. P. Barraca, "Fault-Tolerance in the Scope of Software-Defined Networking (SDN)," *IEEE Access*, vol. 7, pp. 124474-124490, 2019.
- [68] P. Dutta and R. Chatterjee, "A Novel Solution for Controller Based Software Defined Network (SDN)," in *Communications in Computer and Information Science (CCIS)*, vol. 836, pp. 90-98, 2018.
- [69] S. Asadollahi, B. Goswami and M. Sameer, "Ryu Controller's Scalability Experiment on Software Defined Networks," in *2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC)*, 2018.
- [70] Y. Chen, C. Li and K. Wang, "A Fast Converging Mechanism for Load Balancing among SDN Multiple Controllers," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 00682-00687, 2018.
- [71] Y. Zhou, Y. Wang, J. Yu, J. Ba and S. Zhang, "Load Balancing for Multiple Controllers in SDN Based on Switches Group," in *2017 19<sup>th</sup> Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 227-230, 2017.
- [72] J. Yu, Y. Wang, K. Pei, S. Zhang and J. Li, "A Load Balancing Mechanism for multiple SDN Controllers based on Load Informing Strategy," in *2016 18<sup>th</sup> Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2016.

- [73] J. Ansell, W. G. Seah, B. Ng and S. Marshall, "Making Queueing Theory More Palatable to SDN/OpenFlow-based Network Practitioners," in NOMS 2016 – 2016 IEEE/IFIP Network Operations and Management Symposium, 2016.
- [74] G. Nencioni, B. Helvik, A. Gonzalez, P. Heegaard and A. Kamisinski, "Availability Modelling of Software-Defined Backbone Networks," in 2016 46<sup>th</sup> Annual IIE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), 2016.
- [75] B. Xiong, X. Peng and J. Zhao, "A Concise Queueing Model for Controller Performance in Software-Defined Networks," *Journal of Computers*, vol. 11, no. 3, pp. 232-237, 2016.
- [76] P. Kaur and A. Bhandari, "A Comparison of Load Balancing Strategy in Software Defined Networking," *International Journal of Research in Electronics and Computer Engineering*, vol. 6, no. 4, pp. 1018-1025, 2018.
- [77] D. Chourishi, A. Miri, M. Milic and S. Ismaeel, "Role-Based Multiple Controllers for Load Balancing and Security in SDN," in 2015 IEEE Canada International Humanitarian Technology Conference (IHTC2015), 2015.
- [78] K. Benzekki, A. E. Fergougui and A. E. Elalaoui, "Software-defined networking (SDN): a survey," *Security and Communication Networks*, vol. 9, pp. 5803-5833, 2017.
- [79] N. Petroulakis, G. Spanoudakis and I. Askoxylakis, "Fault Tolerance Using an SDN Pattern Framework," in GLOBECOM 2017 - 2017 IEEE Global Communications Conference, 2017.
- [80] A. Banerjee and D. M. A. Hussain, "EXPRL: experience and predication based load balancing strategy for multi-controller software defined networks," *International Journal of Information Technology*, 2020.

- [81] R. K. Das, F. H. Pohrmen, A. K. Maji and G. Saha, "FT-SDN: A Fault-Tolerant Distributed Architecture for Software Defined Network," *Wireless Personal Communications*, vol. 114, pp. 1045-1066, 2020.
- [82] M. Raza, S. Sivakumar, A. Nafarieh and B. Robertson, "A Comparison of Software Defined Network (SDN) Implementation Strategies," *Procedia Computer Science*, vol. 32, pp. 1050-1055, 2014.
- [83] T. Hu, Z. Guo, P. Yi, T. Baker and J. Lan, "Multi-controller Based Software-Defined Networking: A Survey," *IEEE Access*, vol. 6, pp. 15980-15996, 2018.
- [84] F. Kurtz, D. Overbeck, C. Bektas and C. Wietfeld, "Control Plane Fault Tolerance for Resilient Software-Defined Networking based Critical Infrastructure Communications," in *2018 IEEE 5G World Forum (5GWF)*, 2018.
- [85] S. Muhizi, G. Shamshin, A. Muthanna, R. Kirichek, A. Vladyko and A. Koucheryavy, "Analysis and Performance Evaluation of SDN Queue Model," in *International Conference on Wired/Wireless Internet Communication*, 2017.
- [86] T. Issa, Z. Raoul, A. Konate, J. C. Adepo, B. Cousin and A. Olivier, "Analytical Load Balancing Model in Distributed Open Flow Controller System," *Engineering*, vol. 10, no. 12, pp. 863-875, 2018.
- [87] H. Yu, K. Li and H. Qi, "An Active Controller Selection Scheme for Minimizing Packet-In Processing Latency in SDN," *Security and Communication Networks*, vol. 2019, pp. 1-11, 2019.
- [88] J. Cui, Q. Lu, H. Zhong, M. Tian and L. Liu, "A Load-balancing Mechanism for Distributed SDN Control Plane Using Response Time," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1197-1206, 2018.

- [89] W. H. F. Aly, "Generic Controller Adaptive Load Balancing (GCALB) for SDN Networks," *Journal of Computer Networks and Communications*, vol. 2019, pp. 1-9, 2019.
- [90] O. Akanbi, A. Aljaedi and X. Zhou, "Proactive Load Shifting for Distributed SDN Control Plane Architecture," in 2019 16<sup>th</sup> IEEE Annual Consumer Communications & Networking Conference (CCNC), 2019.
- [91] J. Xu, L. Wang, C. Song and Z. Xu, "Minimizing Multi-Controller Deployment Cost in Software-Defined Networking," in 2019 IEEE Symposium on Computers and Communications (ISCC), 2019.
- [92] A. Mahjoubi, O. Zeynalpour, B. Eslami and N. Yazdani, "LBFT: Load Balancing and Fault Tolerance in distributed controllers," in 2019 International Symposium on Networks, Computers and Communications (ISNCC), 2019.
- [93] W. H. F. Aly, "Controller Adaptive Load Balancing for SDN Networks," in 2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN), pp. 514-519, 2019.
- [94] A. Mondal, S. Misra and I. Maity, "Buffer Size Evaluation of OpenFlow Systems in Software-Defined Networks," *IEEE Systems Journal*, vol. 13, no. 2, pp. 1359-1366, 2019.
- [95] M. Escheikh and K. Barkaoui, "Scalable Load Balancing Scheme for Distributed Controllers in Software Defined Data Centers," in 2019 Sixth International Conference on Software Defined System (SDS), 2019.
- [96] B. Lantz and B. O'Connor, "A Mininet-based Virtual Testbed for Distributed SDN Development," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 365-366, 2015.
- [97] B. Lantz, B. Heller and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in 9<sup>th</sup> ACM SIGCOMM Workshop on Hot Topics in Networks, pp. 1-6, 2010.



- [98] “Getting Started - Ryu 4.30 Documentation,” Available: [https://ryu.readthedocs.io/en/latest/getting\\_started.html](https://ryu.readthedocs.io/en/latest/getting_started.html), [Online; accessed April 30, 2020].
- [99] “Ryu SDN Framework,” Available: <https://osrg.github.io/ryu/>, [Online; accessed April 30, 2020].
- [100] L. Kleinrock, *Queueing Systems, Volume 1: Theory*. Wiley-Interscience, 1975.
- [101] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*. Springer, 2012.
- [102] “Mininet,” Available: <http://mininet.org/>, [Online; accessed April 30, 2020].
- [103] L. Mamushiane, J. Mwangama and A. Lysko, “Given a SDN Topology, How Many Controllers are Needed and Where Should They Go?,” in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2018.
- [104] M. K. Faraj, A. Al-Saadi and R. J. Albahadili, “Load Balancing using queue length in SDN based switches,” *Journal of Xi’an University of Architecture & Technology*, vol. XII, no. IV, pp. 2603-2611, 2020.
- [105] A. Mondal, S. Misra and I. Maity, “AMOPE: Performance Analysis of OpenFlow Systems in Software-Defined Networks,” *IEEE Systems Journal*, vol. 14, no. 1, pp. 124-131, 2020.
- [106] M. Hamdan, E. Hassan, A. Abdelaziz, A. Elhigazi, B. Mohammed, S. Khan, A. Vasilakos and M. N. Marsono, “A Comprehensive Survey of Load Balancing Techniques in Software-Defined Network,” *Journal of Network and Computer Applications*, vol. 174, pp. 1-35, 2021.
- [107] S. Gu, J. Kim, Y. Kim, C. Moon and I. Yeom, “Controlled Queue Management in Software-Defined Networks,” in *2015 5<sup>th</sup> International Conference on IT Convergence and Security (ICITCS)*, 2015.

- [108] P. Vizarreta, P. Heegaard, B. Helvik, W. Kellerer and C. M. Machuca, "Characterization of Failure Dynamics in SDN Controllers," in 2017 9<sup>th</sup> International Workshop on Resilient Networks Design and Modeling (RNDM), pp. 1-7, 2017.
- [109] S. Rowshanrad, S. Namvarasl and M. Keshtgari, "A Queue Monitoring System in OpenFlow Software Defined Networks," Journal of Telecommunications and Information Technology, pp. 39-43, 2017.
- [110] G. Huang and H. Y. Youn, "Proactive eviction of flow entry for SDN based on hidden Markov model," Frontiers of Computer Science, vol. 14, no. 4, pp. 1-10, 2020.
- [111] A. Hussein, L. Chadad, N. Adalian, A. Chehab, I. Elhajj and A. Kayssi, "Software-Defined Networking (SDN): the security review," Journal of Cyber Security Technology, vol. 4, no. 1, pp. 1-66, 2020.
- [112] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," in Proceedings of the 2010 internet network management conference on Research on enterprise networking, pp. 3-3, 2010.
- [113] T. Koponen et al., "Onix: A distributed control platform for large-scale production networks," in Proceedings of the 9<sup>th</sup> USENIX Conference on Operating Systems Design and Implementation (OSDI'10), pp. 351-364, 2010.
- [114] R. Sherwood et al., "FlowVisor: A Network Virtualization Layer," Deutsche Telekom Inc. R&D Lab, Stanford, Nicira Networks, Tech. Rep., pp. 1-14, 2009.
- [115] K. Phemius et al., "DISCO: Distributed Multi-domain SDN Controllers," in Network Operations and Management Symposium (NOMS), 2013.
- [116] P. Berde et al., "ONOS: Towards an Open, Distributed SDN OS," in The Workshop on Hot Topics in Software Defined Networking (HotSDN'14), pp. 1-6, 2014.

- [117] Y. Chang et al., “Hydra: Leveraging Functional Slicing for Efficient Distributed SDN Controllers,” CoRR, 2016.
- [118] A. Dixit et al., “ElastiCon; An elastic distributed SDN controller,” in 2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp. 17-27, 2014.
- [119] B. Lee et al., “IRIS: The OpenFlow-based Recursive SDN Controller,” in 16<sup>th</sup> International Conference on Advanced Communication Technology, pp. 1227-1231, 2014.
- [120] S. Hassas et al., “Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications,” in Proceedings of 1<sup>st</sup> Workshop on Hot Topics in Software Defined Networks, pp. 19-24, 2012.
- [121] F. Botelho et al., “SMaRLight: A Practical Fault-Tolerant SDN Controller,” in Proc. 3<sup>rd</sup> European Workshop on Software Defined Network, pp. 6, 2014.
- [122] J. Medved et al., “OpenDaylight: Towards a Model-Driven SDN Controller Architecture,” in Proceedings of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, pp. 1-6, 2014.
- [123] L. Zhu et al., “SDN Controllers: Benchmarking & Performance Evaluation,” Networking and Internet Architecture, vol. arXiv:1902.04491v1, pp. 1-12, 2019.
- [124] S. K Vishwakarma et al., “Performance Analysis of different Load Balancing Algorithms in SDN Based Data Center Networks,” International Journal of Engineering Research & Technology, vol. 11, no. 07, pp. 30-35, 2022.
- [125] O. Belkadi and Y. Laaziz, “A Systematic and Generic Method for Choosing A SDN Controller,” Journal of Computer Networks and Communications, vol. 5, pp. 239-247, 2017.

- [126] L. Girish and S. K N Rao, "Mathematical Tools and Methods for Analysis of SDN: A Comprehensive Survey," in 2016 2<sup>nd</sup> International Conference on Computing and Informatics (IC3I), pp. 774-780, 2016.
- [127] S. Askar and F. Ketik, "Performance Evaluation of Different SDN Controllers: A Review," International Journal of Science and Business, vol. 5, no. 6, pp. 67-80, 2021.
- [128] N. M Kazi et al., "Performance Evaluation of RYU SDN Controller Using Mininet," International Research Journal of Engineering and Technology, vol. 8, no. 10, pp. 892-897, 2021.
- [129] O. M. A. Alssaheli et al., "Software Defined Network based Load Balancing for Network Performance Evaluation," International Journal of Advanced Computer Science and Applications, vol. 13, no. 4, pp. 117-124, 2022.
- [130] M. Elmoslemany, "Performance Analysis in Software Defined Network Controllers," in 15<sup>th</sup> International Conference on Computer Engineering and Systems (ICCES), 2020.
- [131] E. D. Canedo et al., "Performance Evaluation of Software Defined Network Controllers," in Proceedings of the 10<sup>th</sup> International Conference on Cloud Computing and Services Science (CLOSER 2020), pp. 363-370, 2020.



# **List of Publications**

# **LIST OF PUBLICATIONS**

## **PUBLISHED/ACCEPTED/COMMUNICATED**

1. Manmohan Sharma, **Deepjyot Kaur Ryait** and V. K. Saraswat, "Implementation of Software Defined Networks in Internet of Things," *Journal of Emerging Technologies and Innovative Research* (ISSN:2349-5162), vol. 5, no. 12, pp. 469-475, 2018.
2. Manmohan Sharma, **Deepjyot Kaur Ryait** and V. K. Saraswat, "A Comparative Study of Traditional Networks with Software Defined Networks," *Journal of The Gujarat Research Society* (ISSN:0374-8588), vol. 21, no. 6, pp. 483-490, 2019.
3. Manmohan Sharma, **Deepjyot Kaur Ryait** and V. K. Saraswat, "Fault Tolerance Mechanisms in Software Defined Networks," *Journal of The Gujarat Research Society* (ISSN:0374-8588), vol. 21, no. 6, pp. 491-499, 2019.
4. **Deepjyot Kaur Ryait** and Manmohan Sharma, "To eliminate the threat of a Single Point of Failure in the SDN by using the Multiple Controllers," *International Journal of Recent Technology and Engineering* (ISSN: 2277-3878), vol. 9, no. 2, pp. 234-241, 2020.
5. **Deepjyot Kaur Ryait** and Manmohan Sharma, "Implementation of Queuing Models with SDN for Load Balancing in Multiple Controller Environment," *International Journal of Advanced Trends in Computer Science and Engineering* (ISSN: 2278-3091), vol. 9, no. 4, pp. 5872-5879, 2020.
6. **Deepjyot Kaur Ryait** and Manmohan Sharma, "Various Role of the Multiple Controllers in SDN Environment," *International Journal of Advances in Electronics and Computer Science* (ISSN: 2394-2835), vol. 7, no. 12, pp. 17-21, 2020.

7. **Deepjyot Kaur Ryait** and Manmohan Sharma, “Load Balancing between the Multiple SDN Controllers by using Queuing Technique,” *International Journal of Advances in Electronics and Computer Science* (ISSN: 2394-2835), vol. 7, no. 12, pp. 22-26, 2020.



# **List of Conferences**



# **LIST OF CONFERENCES**

## **PUBLISHED/ACCEPTED/COMMUNICATED**

1. **Deepjyot Kaur Ryait** and Manmohan Sharma, “Load Balancing between the Multiple SDN Controllers by Using Queuing Technique,” In International Conference on Recent Advances in Engineering, Technology and Science (ICRAETS), pp. 1-5, 2020.
2. **Deepjyot Kaur Ryait** and Manmohan Sharma, “Various Role of the Multiple Controllers in SDN Environment,” in International Conference on Electrical, Electronics, Computer Science and Information Technology (ICEECSIT), pp. 6-10, 2020.
3. **Deepjyot Kaur Ryait** and Manmohan Sharma, “Significance of Controller in Software Defined Networks,” in 15<sup>th</sup> IEEE International Conference on Industrial and Information Systems (ICIIS-2020), pp. 561-566, 2020.
4. **Deepjyot Kaur Ryait** and Manmohan Sharma, “Load Balancing using Probability Distribution in Software Defined Network,” in International Conference on Advances in Data-driven Computing and Intelligent Systems (ADCIS 2022) Lecture Notes in Networks and Systems Book Series (LNNS) Springer Singapore, vol. 698, pp. 183-200, 2023.
5. **Deepjyot Kaur Ryait** and Manmohan Sharma, “Statistical Influence of Parameters on the Performance of SDN,” in 4<sup>th</sup> International Conference on Communication and Intelligent Systems (ICCIS 2022) Lecture Notes in Networks and Systems Book Series (LNNS) Springer Singapore, vol. 689, pp. 369-384, 2023.
6. **Deepjyot Kaur Ryait** and Manmohan Sharma, “Comparative Analysis of SDN Controllers,” in CODD100 – International Conference on Networks, Intelligence and Computing (ICONIC-2023) is presented and accepted for publication in the Scopus-indexed Taylor & Francis Group 2023.

7. **Deepjyot Kaur Ryait** and Manmohan Sharma, “Performance Evaluation of SDN Controllers,” in 7<sup>th</sup> International Conference on Inventive Communication and Computational Technologies (ICICCT – 2023) is presented and accepted for publication in the Scopus-indexed Springer Lecture Notes in Networks and Systems 2023.
8. **Deepjyot Kaur Ryait** and Manmohan Sharma, “A Novel Approach for Reducing the Single Point Failure in Software Defined Networks,” in 2<sup>nd</sup> International Conference on Recent Advances in Computing Sciences (RACS-2023) is accepted.

