

# LOAD BALANCING IN SOFTWARE DEFINED NETWORKS



By

Sajeela Zainab

00000206428

Supervisor

Asst Prof Mir Yasir Umair, PhD

Department of Electrical Engineering

Military College of Signals (MCS)

National University of Sciences and Technology (NUST)

Islamabad, Pakistan

September 2020

# LOAD BALANCING IN SOFTWARE DEFINED NETWORKS



By

**Sajeela Zainab**

00000206428

Supervisor

**Asst Prof Mir Yasir Umair, PhD**

---

A thesis submitted in conformity with the requirements  
for the degree of Master of Science in

Electrical Engineering

Department of Electrical Engineering

Military College of Signals (MCS)

National University of Sciences and Technology (NUST)

Islamabad, Pakistan

September 2020

# Thesis Acceptance Certificate

Certified that final copy of MS thesis written by **Miss Sajeela Zainab**, Registration No.**00000206428** of Military College of Signals has been vetted by undersigned, found complete in all respect as per NUST Statues/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members of the student have been also incorporated in the said thesis.



Signature: \_\_\_\_\_

Name of Supervisor: **Asst Prof Mir Yasir Umair, PhD**

Date: \_\_\_\_\_

Signature (HOD): \_\_\_\_\_

Date: \_\_\_\_\_

Signature (Dean): \_\_\_\_\_

Date: \_\_\_\_\_

# Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST MCS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST MCS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Sajeela Zainab**

Signature: \_\_\_\_\_

# Declaration

I, **Sajeela Zainab** declare that this thesis titled “**Load Balancing in Software Defined Networks**” and the work presented in it are my own and has been generated by me because of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a Master of Science degree at NUST
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at NUST or any other institution, this has been clearly stated
3. Where I have consulted the published work of others, this is always clearly attributed
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work
5. I have acknowledged all main sources of help
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

**Sajeela Zainab,**

00000206428

# Copyright Notice

- Copyright in text of this thesis rests with the student author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the author and lodged in the Library of MCS, NUST. Details may be obtained by the Librarian. This page must form part of any such copies made. Further copies (by any process) may not be made without the permission (in writing) of the author.
- The ownership of any intellectual property rights which may be described in this thesis is vested in MCS, NUST, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of MCS, which will prescribe the terms and conditions of any such agreement.
- Further information on the conditions under which disclosures and exploitation may take place is available from the Library of MCS, NUST, Islamabad.

# Dedication

This thesis is dedicated to *my beloved parents, teachers and Siblings*

# Abstract

Software Defined Network (SDN) has become a popular standard because of its advantages of centralized controlling and programmability for large-scale networks. In the world of digitization, high network traffic that needs to be handled optimally makes the visualization of whole network extremely complex. Over the years, the amount of traffic flowing through the network has increased exponentially and it is impossible for a single server to handle the huge incoming requests from the client nodes. Load balancing in SDN is used to manage the network traffic by partitioning the load among several servers in order to enhance the overall system efficiency. In this paper, we aim to develop a load balancing approach-using SDN that is essential in World Wide Web for providing high quality service. We thereby propose a Weighted Round Robin scheme that hands out the incoming requests among several servers using three network parameters like amount of data, transmit time and transfer time. The performance of the proposed approach is evaluated and compared with traditional Round Robin load balancing approach using mininet emulation. The results are evaluated on basis of response time (sec), transactions rate (trans/sec) and throughput (Mbit/sec). The results reveal better performance of the proposed scheme as compared to the reference approach.

**Key words:** SDN, Mininet, Datacenter, load-balancing algorithms



# **Acknowledgments**

First, I am thankful to almighty ALLAH who gave me strength and wisdom to complete this work. Moreover, I am also thankful to my supervisor and committee members for their continuous guidance, support and encouragement throughout this research work.

In the end, I am grateful to my parents for their support and prayers.

# Contents

<b>1 Introduction</b> .....	1
1.1 Traditional Architecture: .....	2
1.2 Software-Defined-Networking: .....	2
1.3 Software Defined Architecture: .....	3
1.4 Motivation:.....	5
1.5 Problem Statement: .....	5
1.6 Thesis Outline: .....	5
<b>2 Background and Related Work</b> .....	7
2.1 SDN 3-layer Architecture: .....	7
<b>2.1.1 Architecture Principles of SDN:</b> .....	8
2.2 SDN Application Programming Interfaces (APIs):.....	10
<b>2.2.1 Northbound Interface:</b> .....	10
<b>2.2.2 Southbound Interface:</b> .....	11
2.3 SDN OPENFLOW SWITCH:.....	11
<b>2.3.1 OpenFlow Table:</b> .....	13
<b>2.3.2 OpenFlow Entry:</b> .....	13
<b>2.3.3 Group Table:</b> .....	14
<b>2.3.4 Meter Table:</b> .....	15
<b>2.3.5 OpenFlow Ports:</b> .....	15
<b>2.3.6 OpenFlow Message Types:</b> .....	15
2.4 OpenFlow Connections:.....	17
2.5 OpenFlow Versions:.....	18
2.6 SDN Controllers: .....	19
2.7 Standard Bodies SDN:.....	20
2.8 Load-Balancing:.....	21
<b>2.8.1 Content Aware Load-Balancing Algorithms:</b> .....	22
<b>2.8.2 Content Blind Load Balancing Algorithm:</b> .....	23
2.9 Related Work: .....	23
<b>3 Proposed Methodology</b> .....	27

3.1 Simulation Structure:.....	27
3.2 Network Emulation Tool: MININET .....	27
3.3 SDN based POX Controller:.....	28
3.4 Iperf:.....	28
3.5 Load Analysis Tool: Siege Tool.....	28
<b>3.5.1 Mininet Command:</b> .....	28
3.6 Proposed Topology:.....	29
<b>3.6.1 Mininet Command:</b> .....	29
3.7 Round-Robin Load balancing Technique:.....	30
3.8 Weighted Round-Robin Load Balancing Algorithm: .....	31
3.9 Proposed Method:.....	32
<b>3.9.1 Parameter 1:</b> .....	32
<b>3.9.2 Parameter 2:</b> .....	33
<b>3.9.3 Parameter 3:</b> .....	33
<b>4 Experimental Testbed and Scenarios</b> .....	34
4.1 Case 1: Bandwidth x Delay Product.....	35
<b>4.1.1 Round Robin (r=1):</b> .....	36
<b>4.1.2 Weighted Round Robin (r=1):</b> .....	36
<b>4.1.3 Round Robin (r=5):</b> .....	38
<b>4.1.4 Weighted Round Robin (r=5):</b> .....	39
4.2 Case 2: Transmit Time.....	39
<b>4.2.1 WRR (r=1):</b> .....	40
<b>4.2.2 WRR (r=5):</b> .....	40
4.3 Case 3: Transfer Time .....	41
<b>4.3.1 Weighted Round Robin (r=1):</b> .....	42
<b>4.3.2 Weighted Round Robin (r=5):</b> .....	42
<b>5 Result Analysis and Discussion</b> .....	44
5.1 Case 1: Bandwidth Delay Product (r=1).....	44
<b>5.1.1 Response Time:</b> .....	44
<b>5.1.2 Transaction Rate:</b> .....	45
<b>5.1.3 Throughput:</b> .....	46
5.2 Bandwidth Delay Product (r = 5): .....	47

<b>5.2.1 Response Time:</b> .....	47
<b>5.2.2 Transaction Rate:</b> .....	47
<b>5.2.3 Throughput:</b> .....	48
5.3 Case 2 Transmit Time: ( $r=1$ ) .....	49
<b>5.3.1 Response time:</b> .....	49
<b>5.3.2 Transaction rate:</b> .....	50
<b>5.3.3 Throughput:</b> .....	51
5.4 Transmit time ( $r = 5$ ):.....	51
<b>5.4.1 Response Time:</b> .....	51
<b>5.4.2 Transaction Rate:</b> .....	52
<b>5.4.3 Throughput:</b> .....	53
5.5 Case 3 Transfer Time: ( $r=1$ ).....	54
<b>5.5.1 Response Time:</b> .....	54
5.5.2 Transaction Rate:.....	55
<b>5.5.3 Throughput:</b> .....	56
5.6 Case 3 Transfer Time: ( $r=5$ ).....	57
<b>5.6.1 Response Time:</b> .....	57
<b>5.6.2 Transaction Rate:</b> .....	58
<b>5.6.3 Throughput:</b> .....	59
6 Conclusion and Future Work .....	61
References.....	62

# List of Figures:

Figure: 1.1: Traditional Switch .....	4
Figure 1.2: SDN Basic Design.....	4
Figure 2.1 SDN 3-layer Architecture .....	9
Figure 2.2: SDN OpenFlow Switch .....	12
Figure 2.3: OpenFlow Flow Table Entry .....	14
Figure 3.1: Mininet Command .....	29
Figure 3.2: Server Load Balancing Setup.....	30
Figure 4.1: Proposed Topology in Mininet.....	35
Figure 5.1: Response Time in relation to number of Arriving clients.....	45
Figure 5.2: Transaction rate in relation to number of arriving Clients .....	46
Figure 5.3: Throughput in relation to number of Arriving Clients .....	46
Figure 5.4: Response time in relation to number of Arriving Clients .....	47
Figure 5.5: Transaction Rate in relation to number of Arriving Clients .....	48
Figure 5.6: Throughput in relation to number of Arriving Client.....	49
Figure 5.7: Response time in relation to number of Arriving Clients.....	50
Figure 5.8: Transaction rate in relation to number of Arriving Clients.....	50
Figure 5.9: Throughput in relation to number of Arriving Clients.....	51
Figure 5.10: Response time in relation to number of Arriving Clients.....	52
Figure 5.11: Transaction rate in relation to number of Arriving Clients.....	53
Figure 5.12: Throughput in relation to number of Arriving Clients.....	54
Figure 5.13: Response time in relation to number of Arriving Clients.....	55
Figure 5.14: Transaction rate in relation to number of Arriving Clients.....	56
Figure 5.15: throughput in relation to number of Arriving Clients .....	57
Figure 5.16: Response time in relation to number of Arriving Clients.....	58
Figure 5.17: Transaction rate in relation to number of Arriving Clients.....	58
Figure 5.18: Throughput in relation to number of Arriving Clients.....	59

# List of Tables:

Table 2.2: SDN Controller Comparison .....	20
Table 2.3: SDN Standard Bodies [17].....	20
Table 3.1: Simulation Setup.....	27
Table 4.1: Round Robin for $r = 1$ .....	36
Table 4.2: Bandwidth x Round Trip Time .....	37
Table 4.3: Amount of Data .....	37
Table 4.4: weighted Round Robin for $r = 1$ .....	38
Table 4.5: Round Robin for $r = 5$ .....	38
Table 4.6: Weighted Round Robin for $r = 5$ .....	39
Table 4.7: Transmit Time (ms) .....	40
Table 4.8: Weighted Round Robin for $r = 1$ .....	40
Table 4.9: Weighted Round robin for $r = 5$ .....	41
Table 4.10: Transfer Time.....	41
Table 4.11: Weighted Round Robin for $r = 1$ .....	42
Table 4.12: Weighted Round Robin for $r = 5$ .....	43
Table 5.1: Result analysis using Round Robin and Weighted Round Robin .....	54

# ABBREVIATIONS

SDN: Software Defined Networks

RR: Round Robin

WRR: Weighted Round Robin

Rtt: Round trip time

DCN: Data Center Network

QoS: Quality of Service

API: Application Programming Interface

CLI: Command Line Interface

# Chapter 1

## Introduction

Now a day, a remarkable increase has been occurring in the network traffic. The excessive increase in traffic is due to the immense use of online web applications and different cloud services [1]. Different network operators and vendors need to handle large data setups and online services while maintaining security, quality of service and network availability without increasing operational costs. In industry, Software Defined Network (SDN) has come to meet these challenges [2]. SDN makes the networks to respond dynamically according to the need and availability of the resources. Network infrastructure can be defined instantaneously, respond to user requests and applications, more quickly and simply at a lesser cost [3].

In the arena of computer networking, Software-Defined-Networks (SDN) is a perception, which has added the interest of researchers. It provides an approach to the networking community for programming by simplifying the designing of the networks as compared to the present approach [4]. The main purpose of SDN is to improve the network's reliable communication and management by distinct the forwarding devices (switches or routers) from the control and management planes within a network [2]. Traditional networks have static structure and hierarchy. The traditional network organizes in such a way that flow control and routing plane supervise the devices of the network and it makes the network complex and stops the growth and flexibility of the network infrastructure [5]. Over the



past few years, to facilitate the enormous and vast network flow, a surprising development occurs in Data Center Network (DCN). Special considerations have to be taken to handle such data because any interruption in the service or showing unwanted quality of service (QoS), could lead to an immense data loss [6]. Network traffic consists of data-plane and control-plane traffic and most of the load balancer structures deal with data-plane traffic flow, as it is far more than control-plane traffic.

### **1.1 Traditional Architecture:**

In traditional network architecture, each device has two planes, control and data-plane and both the planes combined itself in one device. To fix traffic across the chain of the switches and routers, traditional systems use combined hardware and software [7]. To achieve valuable information about flow control and packet forwarding each device needs to interact with its neighboring device. Traditional equipment is hardware based that are normally vendor specific, proprietary and expensive and makes the system static and complex [5]. In traditional techniques, load balancing is done by a dedicated hardware and it is hard to match features with other networks due to its vendor specific policies.

### **1.2 Software-Defined-Networking:**

Today, in the field of computer networking Software-Defined-Network (SDN) is a perception, which has added the interest of researchers [1]. In order to improve network monitoring and performance, SDN provides fast and easy network configuration and administration [8]. Now-a-days network architecture needs easy and flexible management as compared to traditional architecture, which is static, localized and complex. SDN decouples the data plane from the control plane. In SDN, centralized controller or network operational system, high programmable tasks are managed

separately [9]. It gives the administration the global image of the whole network and allows changes in the network without any disturbance.

The challenges and intricacy in network scalability of data center servers is one of the dynamic factor. The intelligent switching starts from the network edge router that is connected to the Internet all the way to data center servers in the end [9]. Thus, load balancers have appeared as a powerful or great armament to solve the difficulty of imbalanced load among the servers, load balancing comes as an effective technique, therefore improves the throughput of the network and reduces the response time of incoming requests.

### **1.3 Software Defined Architecture:**

In computer networking, SDN is a current concept that helps to simplify the network management and control. Based on the requirement, SDN provides programmatic access to every one so any one could write its own application. SDN separates the control plane functionality from the data plane [10]. To communicate with physical or virtual switch, the controller uses Open Flow protocol. In networking, open flow protocol gives dynamic programmability and user control. This dynamic programmability gives more access to add functionalities like port statistics, match fields and forwarding activities etc. [11]. To enable development of software the main purpose of SDN is to offer the open interfaces that control the network traffic and its connectivity by using network resources. SDN also provides network inspection and traffic variation performed in the network [5]. There are three means of separating control and data planes: firmly centralized, logically centralized and distributed. Fig. 1.1 shows the traditional switch architecture. In first case, data plane devices have only one function to forward the packets. In the second case, devices are

partially functioned and the central controller holds higher functions [6]. In the last case, we have traditional networking architecture in which both planes are compact with each other and they collaborate with each other to fix the traffic across the network.

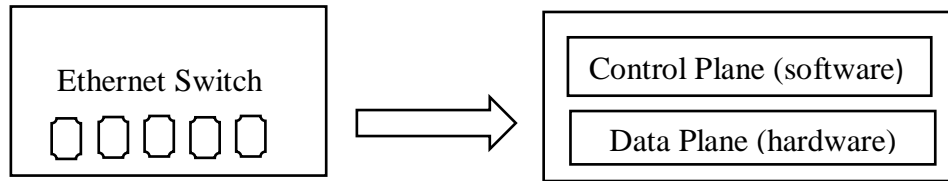


Figure: 1.1: Traditional Switch

To boost the network monitoring as well as network performance, SDN is a modern and new technology in the area of networking, which provides fast configuration and administration of the network.

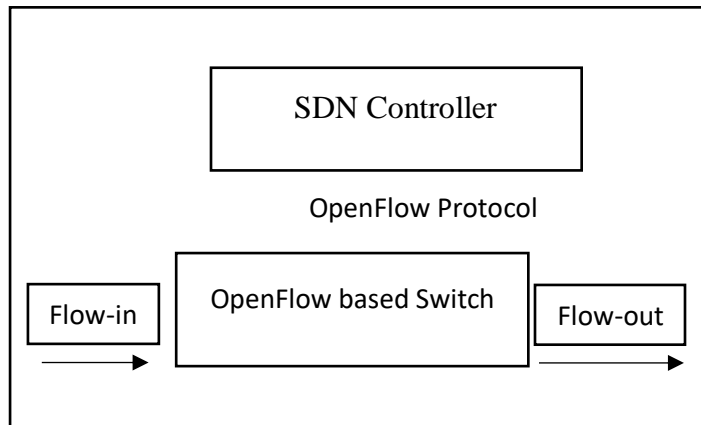


Figure 1.2: SDN Basic Design

Control plane acts as a brain and is capable to manage the traffic by setting different traffic rules in switches for packet processing. With the help of the controller, the administration can easily cope the entire network rather than doing individual configuration on each switch [6]. The only job of data plane is to forward the packets. Fig. 1.2 shows the basic SDN design. Due to its flexibility, we can easily recognize the network topology and allowing flexible and efficient traffic load management by giving priorities to applications and services [12].

## **1.4 Motivation:**

To handle the incoming traffic, most of the web companies use load balancer in their networks. Traditional load balancing is hardware based with the balancing logic loaded on the firmware all the time and is costly to scale. In SDN, software based virtualized devices are defined. Over the same hardware, we can apply different load balancing approaches or algorithms according to the network demand. Motivated by the new paradigm that SDN gives the researchers, a load balancing method aimed as a Software-Defined application over the SDN enabled systems, which is cheaper than hardware based methods.

## **1.5 Problem Statement:**

To achieve better quality of service, network management is essential in a network. In web-based services, servers used to manage and follow up the clients web activities efficiently. To cope a large number of users', a load-balancing application is used to certify better traffic flow. Different balancing methods used to provide reliable communication, internet services and effective resource utilization. In this study, we implement proposed weighted round robin application in an SDN environment and compare its results with already implemented round robin algorithm.

## **1.6 Thesis Outline:**

**Chapter 1** – Overview and objective of the thesis work.

**Chapter 2** – The contextual concepts and SDN technologies defined here that explains its operations and structures. Previous work related to different load balancing methods discussed.

**Chapter 3** – This chapter designates the proposed load balancing technique for the thesis work. .

**Chapter 4** – In this chapter, the software setup described to test the algorithm. We evaluated the experiments and their results also.

**Chapter 5** – The graphical results and discussion defined in this chapter.

**Chapter 6** – It consists of final summary, conclusion with future work

# Chapter 2

## **Background and Related Work**

### **2.1 SDN 3-layer Architecture:**

In SDN 3 layer architecture, control plane and data planes are decoupled, network state and maintenance are centralized logically and from the applications, the underlying network structure is distracted [13]. SDN enables network enterprises to build highly scalable, programmable and flexible networks and gain mechanization and network control according to the business needs.

The SDN infrastructure enables the software to control network connectivity provided by network resources with all possible traffic inspection and modifications. In industry, Software Defined Network (SDN) has come to meet these challenges [14]. SDN makes the networks to respond dynamically according to the need and availability of the resources. Network infrastructure can be defined instantaneously, respond to the user requests and applications more quickly and simply at a lesser cost. The concept of virtualization certifies distract view of the system resources that must be competent to a specific application or a client and inspected by particular clients or applications [15]. The SDN infrastructure supports diverse network functions and connectivity types. It also permits forwarding and processing actions of a network. Network functions may cover physical or virtual and all OSI layers [16]. For SDN to be effective and popular, it must be

capable and deployable within pre-existing multi-layer structures consisting of large organizations.

### **2.1.1 Architecture Principles of SDN:**

The SDN architecture comprises of three basic principles.

- Controller and Data Planes are separable
- Logically centralized control
- Distract network resources and state to peripheral applications.

SDN three-layer architecture consists of Application layer, Control layer, and Data-plane layer. Fig. 2.1 shows the SDN 3-layer architecture.

#### **2.1.1.1 Application-Layer:**

Application-layer is a wide area to develop advanced applications by advantaging all network information about network statistics and topology. It makes the network administrator life easy to use applications to configure and manage underlying environment on top of SDN controller [17]. It consists of different networking features such as forwarding methods, security, network management policies, network configuration and supports the control layer [18]. This layer receives the overall and distant view of the complete network from the controller and provides correct direction to the control layer. Application layer communicates with SDN controller through northbound interface.

#### **2.1.1.2 Control-Layer:**

Control-layer involves of logically centralized SDN controller, which behaves as the brain of the SDN network. The controller comprises of basic logics, control-forwarding devices, manages network traffic flow and policies through Southbound APIs. This is a region

where network vendors come with their own products [19]. To get and maintain different types of network strategies, state information and network topology details, many customized logics are written in SDN controller. It manages the traffic flow control to the underlying switches through Southbound APIs and defines communication to the top layer through Northbound APIs [20]. To communicate with forwarding devices, two protocols are defined by SDN controller i.e. OpenFlow and Open Virtual Switch Database (OVSDB).

### 2.1.1.3 Data-Plane-Layer:

Data plane layer or infrastructure layer also known as forwarding plane. It comprises of different networking devices that forms underlying environment to forward the traffic. In a data center, it could be a set of networking switches and routers [21]. In this layer, the switches and routers forward the incoming and outgoing packets through the control plane. Forwarding plane would be a physical one over which network virtualization takes place through control layer [22].

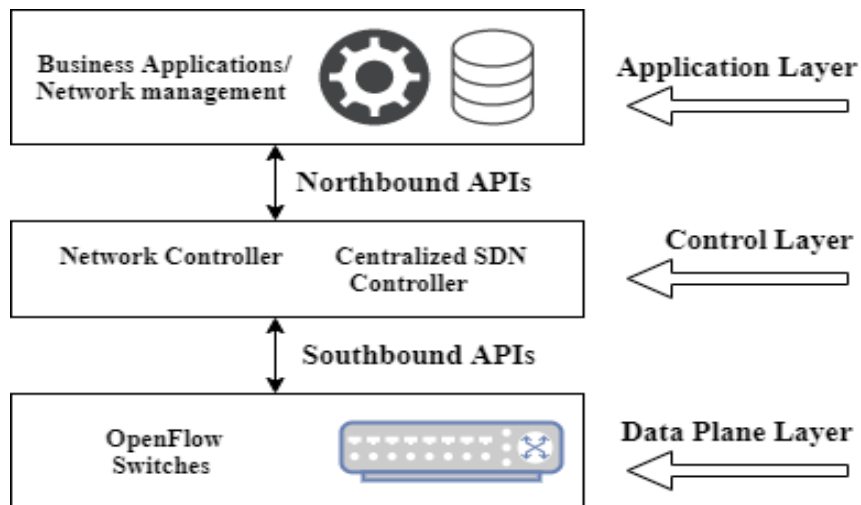


Figure 2.1 SDN 3-layer Architecture



## **2.2 SDN Application Programming Interfaces (APIs):**

There are two important interfaces in SDN environment, one is northbound interface and other one is southbound interface.

### **2.2.1 Northbound Interface:**

Application programming interface such as Northbound-APIs are SDN APIs that provide communication between SDN controller and applications running on top of the network. They provide a link between SDN controller and applications. These APIs facilitate network automation according to the application needs using network programmability [23]. Northbound APIs are most important part of SDN environment as it supports wide variety of applications. The network applications that optimize through northbound interface including firewall, load balancer, security services and cloud services [24]. To automate and simplify the network management and operations, different network programming languages are needed.

The author's emphasize on different important measures of the language for SDN.

- The programming language used for networking must provide the ways to inquire network state and statistics at run time and then deliver it to the applications.
- The programming language should be able to direct network policies for packet forwarding.
- Network applications may define conflicting network rules and policies, to resolve such issues programming language should be powerful to combine different network applications policies [25].

Famous SDN programmable languages that fulfill the network requirements are Pyretic, Freneti, its successor and Procera. These languages direct a clear syntax based on reactive programming functions. These languages define a proper interface for network states and policies due to the reactive nature of programming.

### **2.2.2 Southbound Interface:**

The southbound APIs are OpenFlow (or may be other) protocol in SDN. It defines the communication between SDN controller and underlying environment including physical and virtual switches that determine topology, traffic flows and apply network requests conveyed from Northbound APIs [26]. The main purpose of the southbound interface is to provide connectivity between SDN controller, networking nodes i.e. physical or virtual switches and routers.

Many companies provide a range of services by doing some changes in the programs that are independent of underlying hardware devices [27]. Working with SDN environment, we can control different equipment from a logically centralized control plane, reducing network management and complexity and can easily deploy different services like firewall, security, load balancing, multicasting etc. As business demands shift, the network can easily be rationalized without any interference from the administration [28].

### **2.3 SDN OPENFLOW SWITCH:**

In Software-Defined Networks, OpenFlow switch is a software setup or a hardware equipment that forwards the packets in a network [29]. Either it is based on OpenFlow protocol or any other forwarding protocol suited with it. OpenFlow protocol was first projected at Stanford University by a PhD student who developed a flow-based network called Ethane. When Open Networking Foundation (ONF) was established, a further

development occurred in OpenFlow protocol as Software Defined Networking keystone in 2011.

The OpenFlow switch comprises of one or many flow tables and group table. These flow tables implement packet forwarding and consists of OpenFlow channels to the outward controller [30]. Fig. 2.2 shows OpenFlow based switch. The OpenFlow switch interconnects with the external controller and controller respond to the switch through OpenFlow protocol [31]. By OpenFlow protocol, the SDN controller reactively or proactively adds, updates and deletes flow entries in flow table. In a switch, each flow table comprises of flow entries. Each flow entry involves of counters, match-fields and group of instructions to match a packet.

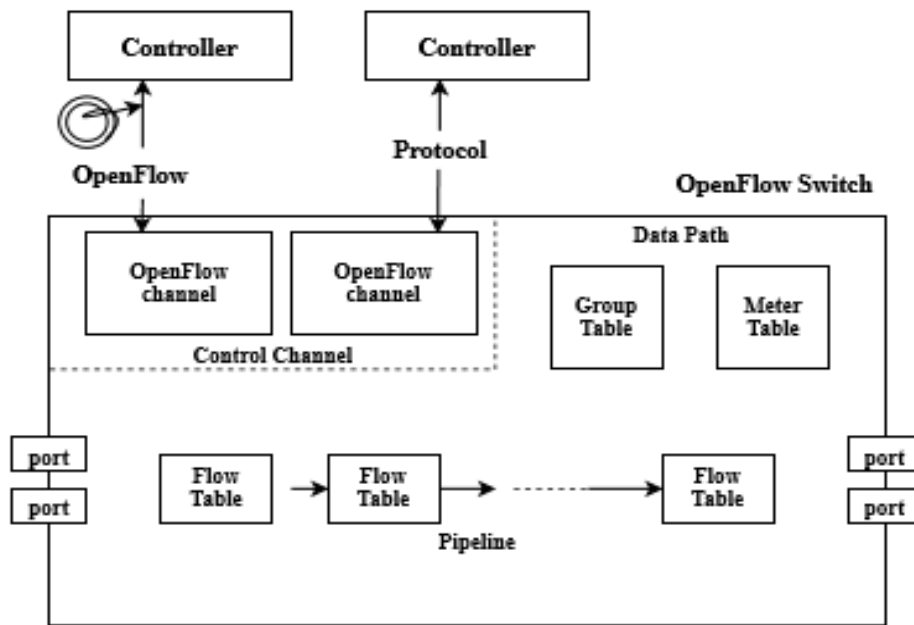


Figure 2.2: SDN OpenFlow Switch

### **2.3.1 OpenFlow Table:**

Flow tables consists of different priority rules. Flow rules comprises of match conditions and action fields. When packet arrives, match field decides which rule applied to the arriving packet and action field defines how to handle a packet [32]. As packet comes, the forwarding device matches the rule to the incoming packet and action that has high priority is chosen and applied to the packet [33]. If no rules defined and matched, then packet directed to the controller in the form of an open flow message. It is called packet-in event, including layer 2, 3 and 4 packet headers also including port information on which packets arrived [34]. To handle a new packet, a controller received this message and a new flow rule added to the flow table.

### **2.3.2 OpenFlow Entry:**

There are three basic aspects of an OpenFlow based flow-entry. To match the packet header, the match rule used for this purpose. User directs the flow rules through controller, inserted in the TCAM. Each packet captures Ethernet, IP and TCP protocol [35]. Flow rules implementation of the controller are handled differently according to the language they are programmed in, like Floodlight and RYU controller. Fig. 2.5 shows open flow table entry.

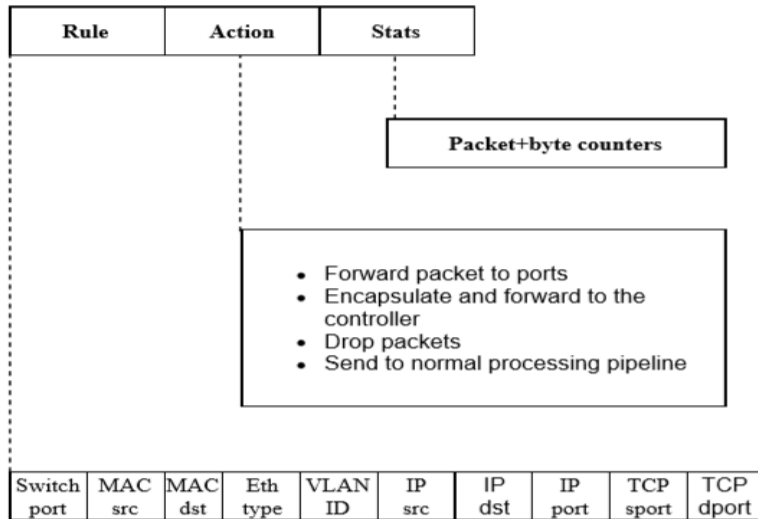


Figure 2.3: OpenFlow Flow Table Entry [15]

When packet matches a rule, the actions are directed as orders by SDN controller that should be followed by OpenFlow switch. OpenFlow protocol also keeps the counters for packet statistics. These statistics retains the number and size of packets passing through the switch. When packet reaches to any port of the switch, it passes through some steps depending on the actions that are preserved in OpenFlow table [16].

A switch may forward or drop the packet according to the rules that are set in flow-table. If the incoming packet does not match to any rule in the flow table than it passed to the controller through OpenFlow based channel, dropped or might forward to the next flow-table [36].

### 2.3.3 Group Table:

Group table defines the complex actions for packet forwarding and flooding such as link aggregation and multipath routing etc. It also supports various flow entries to be forwarded to a single hop and changes the output actions across multiple flow entries.

### **2.3.4 Meter Table:**

In OpenFlow, meters are important for traffic rate monitoring former to the output. The ingress rate of network traffic is easily monitored with meters defined by the flow. Using (goto meter), instruction flows direct the packets to a meter table, where meter performs some actions on basis of received packet rate.

### **2.3.5 OpenFlow Ports:**

OpenFlow ports are important to pass the packets between the network and OpenFlow processing. Using OpenFlow ports, switches are connected logically with each other and forward packets from one switch to the other switch using OpenFlow output port. OpenFlow switch consists of some extra OpenFlow ports because some network interfaces are restricted for OpenFlow networks [37]. In an ingress port, OpenFlow packets are received and managed by the pipeline that forwards the packets to an output port. In OpenFlow switch, ingress port is an OpenFlow port that receives the packet and it is used for matching the packets. Using output function, pipeline decides to direct the packets to an output port [38]. OpenFlow switch also consists of three types of ports: logical, physical and reserved ports.

### **2.3.6 OpenFlow Message Types:**

The OpenFlow entails three sorts of messages, i.e. symmetric, a-synchronous and message from controller to switch.

### 2.3.6.1 Symmetric Message:

Symmetric messages sent both from the controller and switch side. The OpenFlow switch is capable to send and define the following symmetric messages to and from the controller.

- **Hello message:** When connection begins, hello message is exchanged between switch and controller.
- **Echo message:** Echo request, reply message sent from either controller or switch to check or validate the liveness of controller-switch connectivity, and may use to find link bandwidth or latency.
- **Experimenter message:** It offers additional functionalities for future work.
- **Error:** Controller or switch uses the error message to report a connection problem. It is mostly used by OpenFlow switch to direct a request failure introduced by the controller.

### 2.3.6.2 A-Synchronous Message:

The a-synchronous messages are exchanged from switch to the controller [39]. It consists of following types of messages.

- **Packet-In:** This packet-in message is used when arriving packet does not match any rule set in flow table and switch buffering status.
- **Flow-Removed:** This message informs the controller for the removal of flow-entry from the flow-table.
- **Port-Status:** This message is used during port configuration settings and sends switch port status to the controller.

- **Controller-Status:** When the OpenFlow channel status changes, this message is sent to the controller.
- **Flow-monitor:** If any change occurs in flow table, this message informs the controller about the change and controller defines some rules to tackle changes in a flow table.

### **2.3.6.3 Controller to Switch:**

Controller to switch message consists of messages that controller sends directly to the switch to achieve its network state. It consists of following messages.

- The controller knows the functionalities of the switch and it reacts with the feature reply.
- To change the configuration parameters of the switch, controller used configuration type message.
- Controller uses this packet-out message to direct packets out of port on OpenFlow switch. This message comprises of different actions that must be functional in a distinct order.

## **2.4 OpenFlow Connections:**

To exchange the OpenFlow messages between OpenFlow based switch and controller, OpenFlow channel is used. A single controller accomplishes various OpenFlow channels each to a different OpenFlow based switch [40]. Each switch has one channel to a single OpenFlow controller and for reliability may have various channels each to a different controller.

A controller directs various OpenFlow channels concurrently to numerous switches. It may also define many channels to a single OpenFlow switch as well. Using a standard



TLS or TCP connection, an OpenFlow switch connects to a controller at a fixed TCP or IP port 6633. The OpenFlow switch must distinguish the message set in OpenFlow channel and internet traffic channel [41]. When a connection is defined, an OFPT\_HELLO message having OpenFlow version is interchanged between switch and controller. The only condition for the network that it would deliver TCP/IP connection is that if any disruption occurs between switch and controller connection, the switch should define one of the modes [42]. It could drop the packets for OpenFlow controller, or it may act as a legacy switch.

## **2.5 OpenFlow Versions:**

In a secure channel, OpenFlow protocol is used between a switch and a controller. OpenFlow protocol is constantly upgraded with new versions or extensions [43]. During ONF normalization process, OpenFlow protocol is first developed from version 1.0. In first version, there is one single flow table and maximum 12 match fields; whereas latest version of protocol supports multiple flow tables and number of novel functions as well and according to the demand, the competency and scalability of version is extended with time [44]. Table 2.1 show version comparison. In February 2011, OpenFlow protocol version 1.1 was released and Open Networking Foundation (ONF) set improvement of the standard. In December 2011, Open Networking Foundation (ONF) officially announced version 1.2 and distributed it in February 2012. The version 1.5.1 is the latest version of protocol.

Table 2.1: OpenFlow Version Comparison [45]

Version	Major Feature	Reason	Use Cases
1.0 - 1.1	Multiple table	Avoid flow entry explosion	
	Group Table	Enable Applying action sets to group of flows	Load balancing, Failover, Link Aggregation
	Full VLAN and MPLS Support		
1.1 - 1.2	OXM Match	Extend matching flexibility	
	Multiple Controller	HA/Load balancing/Scalability	Controller Failover, Controller Load Balancing
1.2 - 1.3	Meter table	Add QoS and DiffServ capability	
	Table miss entry	Provide flexibility	
1.3 - 1.4	Synchronized Table	Enhance table scalability	Mac Learning/Forwarding
	Bundle	Enhance switch synchronization	Multiple switch configuration
1.4 - 1.5	Egress Table	Enabling processing to be done in output port	
	Scheduled bundle	Further enhance switch synchronization	

## 2.6 SDN Controllers:

The controllers in SDN environment are software based and programming language is most important for application development and controller performance [46]. The authors state that Java is a good option as it supports multithreading. At performance level, python has issues with multithreading, C and C++ have issues with memory. A number of open source controllers are introduced in SDN environment. The SDN controller acts as a brain or central part that uses OpenFlow protocol to cooperate with data plane. The controllers are inscribed in software, providing elasticity to manage and control OpenFlow tables [47]. The flow tables could be injected with rules proactively (before packet arrives) or reactively (after packet arrives). Every controller comes with its GUI and REST API interface for the connectivity with graphical user interface and predefined command line interface to set and enable routes through OpenFlow protocol. Table 2.2 shows SDN controller comparison.

Table 2.2: SDN Controller Comparison

Name	Developer	Language	Learning Curve
RYU	Ryu community	Python	Medium
OpenDayLight	ONF	Java	High
Floodlight	Big Switch Network	Java	Medium
Pox	Nicira	Python	Low
Nox	Nicira	C++	Low

The first controller that was developed by Stanford University is **NOX**. It is written in C++ language and licensed under GNU GPL (General Public License). The **POX** controller is the up version of NOX written in python language and could be used in various SDN applications [47]. POX controller is commonly used for educational or academic purpose to introduce or interact the students with SDN knowledge and it is available under APL license (Apache Public license). The **Beacon** controller is written in Java language and introduced at Stanford University. **Floodlight** controller is advanced version of Beacon controller and it is licensed under APL [45]. **OpenDayLight** controller is also inscribed in Java programming language and presented by Linux. It is issued under EPL (Eclipse Public License) and has no limitation on operating system. **RYU** controller is the upcoming controller.

## 2.7 Standard Bodies SDN:

Many standard developing organizations (SDO) and development enterprises are involved in forming standards and guiding principle for SDN. Table 2.3 defines central SDO and other organizations involved in developing the standards.

Table 2.3: SDN Standard Bodies [48]

<b>Organizations</b>	<b>Task</b>	<b>Effort related to SDN</b>
Open Networking Foundation (ONF)	An industrial group committed to upgradation and adoption of SDN via open standard development.	OpenFlow
Internet Engineering Task Force (IETF)	It is an internet technical standard and defines RFCs and standards related to internet.	Interface to routing systems (I2RS) Service function chaining
European Telecommunications Standards Institute (ETSI)	ETSI defines worldwide applicable standards for network communication and information technologies.	NFV

### **2.8 Load-Balancing:**

Load balancing is a way of distributing or allocating the network load to each distinct server in a cluster to improve the throughput, response time and resource utilization in an effective manner. Simultaneously, it takes away a situation in which some of the data center servers are highly loaded or others are lightly loaded. Different load-balancing algorithms are used to balance the client requests across a group of servers.

The chief purpose of load balancer is to direct the incoming data in such a way that not only a single server receives all the requests and starts working slowly, thus affecting the performance. Various parameters are important for load balancing methods. One of them is content awareness. If load balancing application checks and concerns what type of

information is passing through its channel it is considered as content aware otherwise it is content blind.

### **2.8.1 Content Aware Load-Balancing Algorithms:**

The load-balancing algorithm could be developed to accommodate different types of data such as online gaming, web loading, banking etc.

#### **2.8.1.1 IP Hash Technique:**

The IP Hash load-balancing algorithm combines the source and destination IP address in such a way to construct a new hash key. The generated key is used to direct the client requests to a server. If a connection breaks up, the key is regenerated and client's request is sent to the same server.

#### **2.8.1.2 DNS Technique:**

For a particular domain, DNS based load balancing helps to improve client requests. It consists of multiple techniques to allocate, retransmit or define the load balancing process. One method is DNS delegation. A sub domain is delegated to zones that are functioned by set of servers serving a web site. By doing so the website will be able to tackle and distribute the requests in a better way.

#### **2.8.1.3 Persistence Technique:**

This type of load balancing is mainly used in companies where HTTP connections would not be enough like banks or security concerns organizations. A sticky session is created between client and server till the connection lasts. Using source and destination IP addresses, data traffic is passed through the channel unless session ends.

## **2.8.2 Content Blind Load Balancing Algorithm:**

Content blind techniques could be operational and inexpensive if data traffic passing through network channel is in a general form. It does not provide a persistent path but delivers requests in high speed and may respond to a new request in a faster way as well.

### **2.8.2.1 Latency or Weighted Response Time:**

This load-balancing algorithm is used mostly in online gaming. The server, which gives the fastest response time, has high priority to serve the requests first. Therefore, there is no delay or gap shown by the application during online gaming.

### **2.8.2.2 Round-Robin or Weighted Round Robin:**

A round-robin algorithm is a simple technique where the IP address of servers is used in a circular or rotational manner. The algorithm could be modified by assigning weights to a server. Server with higher priority has greater weight with respect to other network parameters and more requests are sent to server with greater weight.

### **2.8.2.3 Adaptive or Dynamic Technique:**

In adaptive or dynamic load balancer, an agent is installed in each server that reports its current status to the load balancer. Therefore, the load balancer uses existing values or historical data with already implemented techniques or may use some other inventive technique to handle the incoming requests.

## **2.9 Related Work:**

In SDN domain, several load-balancing techniques are suggested, but still it is an open challenge for academia researchers and industry. OpenFlow protocol permits the network

developer to use different programming languages that are compatible with SDN environment and allows the use of different software applications and methods that was not possible in earlier traditional network architecture. Static and dynamic load balancing algorithms are defined in SDN domain on basis of networking parameters. For smaller networks, static load balancing algorithm is used. Performance analysis of static method is good with respect to dynamic load balancing method. In huge web server fields, static is not moral where client requests are generated enthusiastically. Dynamic load-balancing applications are used for wide-reaching large web networks as it sets the requests on basis of current state of the server.

In [49], server-based network, dynamic load-balancing algorithm is proposed. The algorithm selects the finest server because of different parameters like server memory utilization, CPU utilization and number of available connections per server. On basis of these parameters' server capacity is calculated and client requests are furthered to the least loaded server. The parameters used for comparison are usage of CPU, server memory usage, throughput and response time with random and round-robin load-balancing techniques.

In [50], they did SDN based load balancing by examining and evaluating different load balancing strategies. They have implemented round-robin, weighted round-robin algorithm, flow statistic algorithm, and gather all information like response time, throughput and transactions time. By considering the above-mentioned parameters, we have to find which algorithm is efficient to balance the load effectually.

Referring [51], a low cost load-balancing framework (L2RM) is introduced that improves the fat-tree data center topology dynamically. The L2RM consists of three main aspects.

First, they calculate load deviation parameter to measure the path congestion. If congestion occurs, then second algorithm that is ARM (adaptive route management) scheme is applied to select the alternative path and balance the link load. For controller overhead, a DIP (dynamic information polling) scheme is applied to lessen the controller overhead by keeping exact and correct information of the switches.

Stating [52], a packet based load balancing algorithm is implemented using ONOS controller. First, the network topology is discovered using LLDP protocol. Then Dijkstra, shortest-path algorithm is used to find shortest path between two hosts. A threshold value is defined based on average traffic passing through the link and processing capacity of the link. If network traffic is less than the threshold value then it follows the default path but if arriving packets are higher than threshold value then traffic is redirect to the other core switch or to the alternative link without any packet loss.

In [53], SDN based load balancing method is used to implement multi controller load balancing and compare their results with traditional network. In this paper, two scenarios are implemented first scenario is with single controller that is responsible to program the OpenFlow switches for traffic forwarding. In other scenario, the second controller is responsible for load balancing application. At the end, the delay and throughput results are compared with traditional network.

Referring [54], an adaptive SDN based load-balancing algorithm is explained. A DCN fat-tree topology is considered using bandwidth and loss parameters. When the network throughput is reduced below its threshold value or if there is an increase in traffic, there is an increase in loss as well than its expected value. The proposed algorithm works and



finds an optimal path considering loss and bandwidth parameter and compare the results with traditional approach.

In [55], a dynamic load-balancing approach for data center network topology is presented. In this scheme, load balancing is done at a given time by computing the transmission costs of the links. Dijkstra shortest-path technique is used to find multi paths between source and destination hosts. The path with least cost and load is selected and traffic is forwarded to that optimal path and switch-forwarding table is updated with time. For comparison, bandwidth and delay results are compared for before and after load balancing.

# Chapter 3

## Proposed Methodology

In this chapter, we discuss the simulation structure, SDN based Controller, our network topology, traffic generator tool, simulation results and its observations.

### 3.1 Simulation Structure:

The simulation setup has designed with PC (laptop):

Table 3.1: Simulation Setup

Processor:	Intel Core i5
System type:	64-bit operating system
Oracle	Virtual Machine
Ubuntu	16.04
Mininet	Installed Mininet on Ubuntu 16.04
POX Controller	Install controller on Ubuntu 16.04

### 3.2 Network Emulation Tool: MININET

Mininet is a network emulation open source tool. The tool is capable of creating virtual environment having virtual hosts, switches and application code is usually running on top of SDN based controller [56]. The setup is programmed using python programming language and using command line interface (CLI) user can easily connect with the network simulation.

### **3.3 SDN based POX Controller:**

**POX** is SDN based OpenFlow controller written in python language. To forward the packets among different hosts, SDN controller is required to install flow entries to the OpenFlow switch. The POX controller is installed inside virtual machine based Mininet tool [57]. POX is an open source controller, that allows developing and running different SDN based applications like hub, firewall, multicasting and load balancing etc.

### **3.4 Iperf:**

To measure the performance of a network, Iperf is commonly used tool for tuning. It is an open source tool written in C language and runs on numerous platforms like Windows and Linux etc. Iperf consists of client/server functionality [58]. It measures the throughput by generating data streams between two hosts i.e. sender and receiver.

### **3.5 Load Analysis Tool: Siege Tool**

Siege is a load-testing tool used to find the performance parameters of a web server. Using siege tool, we can find out the throughput, response time of a server, data transfer rate, transaction rate and many other parameters that are important to measure network performance [59]. The traffic load generated by siege tool is equal to the number of concurrent users (-c), multiplied with number of request (-r) by each client or user.

#### **3.5.1 Mininet Command:**

```
ubuntu@sdnhubvm:~[01:15]$ cd siege
```

```
.siege
```

```
ubuntu@sdnhubvm:~/siege[01:15]$ siege 10.0.0.1 -c10 -r1 -t5s
```

-c: number of concurrent users

-r: number of user requests.

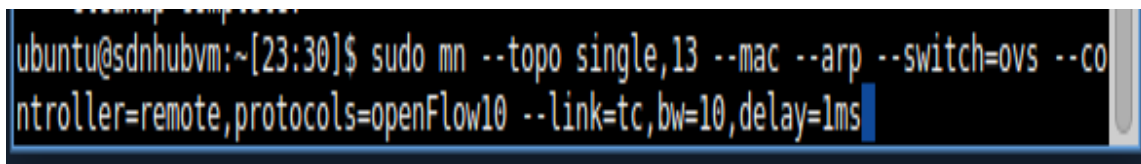
By increasing number of concurrent users, number of requests also increases.

### 3.6 Proposed Topology:

A topology is created in Mininet graphical interface Miniedit consisting of servers and hosts on opposite side of the switch. In our proposed topology, there are 10 hosts, 3 servers, an OpenFlow switch and a POX controller. The hosts 1, 2 and 3 will act as a web server. The host 4 to host 13 act as a clients and each client will generate different requests. On host 4 to host 13, a siege tool is installed and run to generate the network traffic.

#### 3.6.1 Mininet Command:

```
sudo mn --topo single,13 --mac --arp --switch=ovs --controller=remote,protocols=openFlow10 --link=tc,bw=10,delay=1ms.
```

A terminal window screenshot showing the execution of the Mininet command. The prompt is 'ubuntu@sdnhubvm:~[23:30]\$' and the command entered is 'sudo mn --topo single,13 --mac --arp --switch=ovs --controller=remote,protocols=openFlow10 --link=tc,bw=10,delay=1ms'. The cursor is at the end of the command line.

```
ubuntu@sdnhubvm:~[23:30]$ sudo mn --topo single,13 --mac --arp --switch=ovs --controller=remote,protocols=openFlow10 --link=tc,bw=10,delay=1ms
```

Figure 3.1: Mininet Command

In server load balancing approach, many modules are involved such as virtual IP (VIP), clients and server machines. Figure 3.2 shows server load balancing setup in SDN environment.

#### VIP:

Virtual IP (VIP) is an IP address, which is publicly accessible to the world to get the web browsing. A Virtual IP is not bound to any physical interface. A TCP or UDP port is linked with it. For example, in TCP port 80 is used for web traffic. There may be a single server or multiple servers associated with VIP hence virtual IP distributes the load among servers using different balancing techniques.

## Servers:

A server may be defined as HTTP server or it may be one of the many other services associated with it. Each server has an IP address, TCP or UDP port is linked with it and it is not accessible publicly.

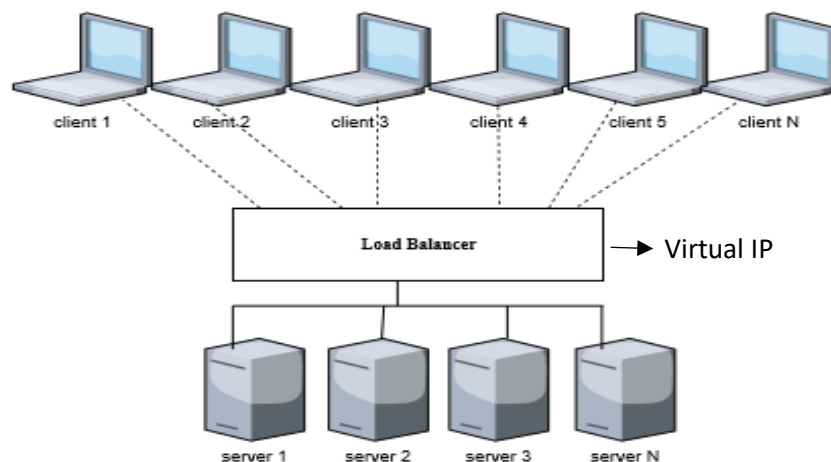


Figure 3.2: Server Load Balancing Setup

### 3.7 Round-Robin Load balancing Technique:

In data center, Round-Robin method is used for server load balancing. It passes the requests to server in circular manner and shares the client requests to a group of servers. The main benefit of round robin load balancing technique is that it is simple to implement,

low-cost and works good when all the servers have same capability or capacity [60]. In OpenFlow based SDN environment, firstly the controller is modified and statistical information of all servers like IP address, MAC, port number or any other information are composed and stored in a Hash map. To assign the incoming requests, virtual IP (VIP) component calls OpenFlow based round-robin algorithm, the algorithm finds which server is to be used on basis of last selected server id, and it guarantees that all servers work in a circular manner.

### **3.8 Weighted Round-Robin Load Balancing Algorithm:**

Weighted round robin is widely used method for server load balancing applications. The algorithm allows user requests on basis of weight assigned to each server. The servers are assigned weights according to the different performing parameters [61]. If server 1 has capacity 3 times more than server 2, then weight of 3 is assigned to server 1 and weight of 1 to server 2.

#### **Pseudo Code:**

```
\begin{algorithm} {Weighted Round Robin Algorithm}
{Input}:
{Arriving\_Requests}
{Output}:
{{Server\_Select}}
{Start}:
{Arriving\_Requests}
Calculate current load of each server using above mentioned methods
while do
traverse all the servers
choose a least loaded server node with highest weight
def choose server node
best weight = 0
best server node = None
for server node in n nodes
```

```

effective_weight = server node weight
if effective_weight <= 0:
best_node = node
break
if effective_weight >= best_weight:
best_weight = effective_weight
best_node = node
return best_node
EndIf
EndFor
EndProcedure
end{algorithm}

```

### 3.9 Proposed Method:

The proposed method forwards the Software Defined load balancer that selects server on basis of different parameters.

In round-robin load balancing, the algorithm balances incoming client requests in a circular manner without considering any performance parameter.

In our proposed method, we will implement weighted round robin algorithm by considering three different performance parameters. According to these parameters, we assign the weights to each server and compare results.

For the space of thesis, following parameters are considered.

#### 3.9.1 Parameter 1:

When establishing high-performance networks, bandwidth x delay product is significant to know as it relates to the number of bits transmitted by sender to the receiver. Data transmitted over the media can be calculated by following formula:

$$\text{Amount of data} = \text{maximum Bandwidth (Mb/sec)} \times \text{round\_trip\_time (ms)}$$

Most commonly for network performance we are concerned with RTT state, that is delay in general words so amount of data transferred can be simply written as bandwidth x

delay product. Through bandwidth delay product, we find out whether our network is long fat network (LFN) or not. The basic criteria is if bandwidth delay product is more than  $10^5$  bits then network is LFN otherwise it is not.

### 3.9.2 Parameter 2:

The second parameter being considered is time interval between starting and completion of transfer of a file. It also includes the additional time spent on setting or requesting transfer of file through network.

$$\text{Transfer time} = \text{Round trip time (ms)} + 1 / (\text{bandwidth (Mbps)} \times \text{transfer size (MB)})$$

### 3.9.3 Parameter 3:

Third parameter in observance is transmission time of data. The transmit time is amount of time taken from the first bit up to the last bit of a file or message to leave the transmission node.

$$\text{Transmit time} = \text{Transfer size} / \text{transfer time}$$

Using these parameters, we find least loaded server for traffic load balancing. The proposed weighted round robin algorithm is aimed to handle the servers in a better way with different processing capacities. The assigned weight of each server defines its processing capability. The server with high weight receives more requests than other servers. The proposed algorithm consists of 'n' servers set  $s_0, s_1, s_{n-1}$ .  $W(S_j)$  defines the weight of server  $S_j$ , j indicates last selected server and it is initialized with -1. EW is the effective weight and it is defined as zero and  $\max(S)$  is the maximum weight of all servers in S.



# Chapter 4

## **Experimental Testbed and Scenarios**

In this chapter, performance evaluation of the proposed weighted round robin load balancer is main objective. It uses multiple parameters to decide which server handles the incoming requests. Using HTTP connections, we have compared the response time, transaction rate and throughput against round robin load balancer by increasing users or incoming requests. Fig. 4.1 shows the graphical view of topology. The topology consists of 10 hosts, each host consists of different number of clients, and each client generate different traffic load. The IP addresses of hosts from 10.0.0.4 to 10.0.0.13 respectively as clients and the hosts with IP addresses 10.0.0.1, 10.0.0.2 and 10.0.0.3 act as servers. The server IP addresses are not visible to clients. The load balancer acts as a central identity that hides identity of servers after it and parodies the identity of servers. A load balancer is identified with virtual IP address and MAC address that is not used for any other device in the network. In the network topology, all clients connected to this virtual IP and MAC address, believe that it is the web server address of server that is answering its requests.

In the topology, switch is defined between clients and servers and is connected to a POX controller through OpenFlow protocol having default port 6633. The POX controller is running with load balancer application for each new experiment.

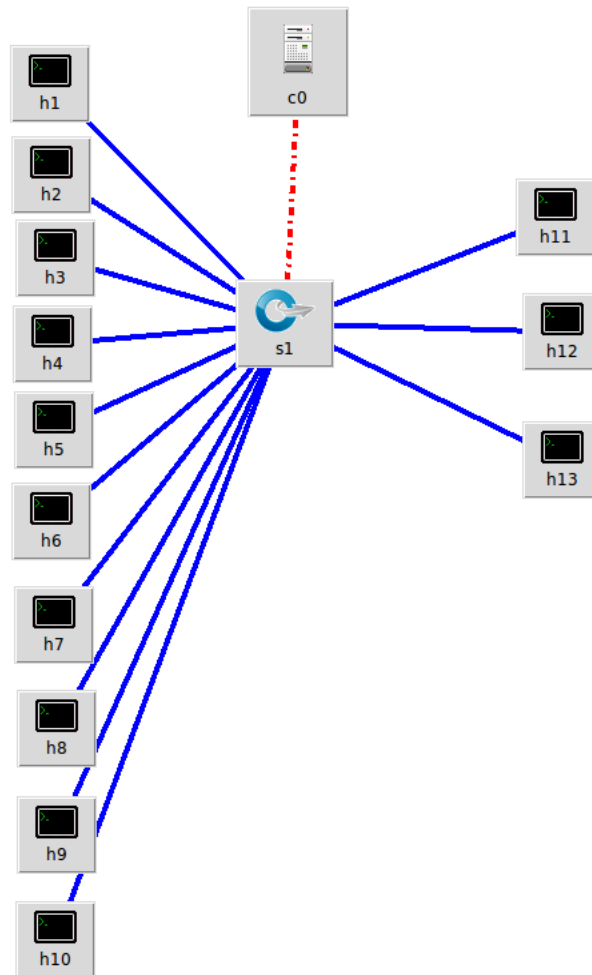


Figure 4.1: Proposed Topology in Mininet

## 4.1 Case 1: Bandwidth x Delay Product

In our first case, both Round-Robin and weighted Round-Robin load balancer is subjected to client requests ranging from 10 to 500 connections. All the client requests are sent to load balancer IP address. The application running for load balancer, handles the packets

in controller and then passes them to OpenFlow switch that forwards it to any one IP in the pool of servers.

#### 4.1.1 Round Robin (r=1):

First, we implement round robin load-balancing algorithm for different number of users using siege tool. The method uses a counter to pick up a server. When first request arrives, it is forwarded to the first server, the second request is sent to second server and third request to third server. The next request is sent to first server again as it is now first in queue. Using siege tool, we set concurrent users (-c) from 10 to 100 with (-r=1) gives total transactions. Table 4.1 shows the results by taking corresponding number of users.

Table 4.1: Round Robin for r = 1

No of users	Throughput (Mbit/s)	Response time (Secs)	Transaction rate (Trans/sec)
10	0.13	0.22	41.01
20	0.12	0.50	37.12
30	0.09	0.76	27.08
40	0.07	1.35	22.18
50	0.03	1.72	14.01
60	0.05	1.71	9.64
70	0.03	1.48	8.70
80	0.04	1.62	12.47
90	0.03	2.10	11.72
100	0.04	2.45	9.00

#### 4.1.2 Weighted Round Robin (r=1):

For our proposed method, we find bandwidth delay product parameter. Using Iperf, we find the maximum utilized bandwidth of each link in Mbit. Using ping command, we generate a traffic and find its round\_trip\_time (rtt) in milliseconds (ms). Table 4.2 shows the network bandwidth and rtt values. After getting bandwidth and delay values, we assigned weight to each server and run the weighted round robin application.

Table 4.2: Bandwidth x Round Trip Time

No. of Users	Max Bandwidth (Mbit/sec)	RTT (ms)
10	9.58	4.403
20	9.83	5.786
30	10	4.302
40	6.43	5.358
50	5.38	4.547
60	5.12	4.280
70	8.57	5.026
80	10	4.862
90	9.86	4.989
100	5.43	4.423

From bandwidth and rtt values, we find bandwidth delay product and get amount of data in the form of bits. Table 4.3 shows the received amount of data by each server. The server receiving least number of bits is least loaded server and higher weight is assigned to that server.

Table 4.3: Amount of Data

Servers	Amount of data (bits)
Server 1	$3.7664 \times 10^4$ bits
Server 2	$4.2562 \times 10^4$ bits
Server 3	$3.6030 \times 10^4$ bits

After getting average amount of data each server is handling, we assign a higher weight to server 3 as it is least loaded server, then server 1 is assigned weight at second priority and at last least weighted server is server 2. Table 4.4 shows the results after running WRR application.

Table 4.4: weighted Round Robin for r = 1

Users	Throughput (Mbit/sec)	Response time (Secs)	Transaction rate (Trans/sec)
10	0.33	0.11	88.21
20	0.32	0.20	85.58
30	0.32	0.26	84.29
40	0.31	0.34	84.18
50	0.31	0.46	82.20
60	0.29	0.49	77.15
70	0.25	0.57	66.59
80	0.23	0.72	62.66
90	0.23	0.74	62.03
100	0.21	0.79	56.24

#### 4.1.3 Round Robin (r=5):

In this test, using siege tool we increase the client requests or load by setting number of concurrent users (-c) from 10 to 100 with -r equal to '5'. For example, if we have 10 simultaneous users when r is taken as five, then total requests or load equals to 50.

Table 4.5: Round Robin for r = 5

No. of Users	Throughput (Mb/sec)	Response time (Secs)	Transaction rate (Trans/sec)
10	0.14	0.22	43.17
20	0.09	0.57	28.05
30	0.10	0.79	28.83
40	0.10	0.84	30.22
50	0.05	0.81	15.13
60	0.051	1.11	14.38
70	0.06	0.58	17.32
80	0.08	0.95	23.89
90	0.03	0.78	10.18
100	0.08	0.98	23.67

#### 4.1.4 Weighted Round Robin (r=5):

'r' is set to five for WRR as well, so increasing the average load or traffic and deriving results.

Table 4.6: Weighted Round Robin for r = 5

No. of Users	Throughput (Mbit/sec)	Response time (Sec)	Transaction rate (Trans/sec)
10	0.24	0.15	63.89
20	0.22	0.29	58.53
30	0.22	0.32	57.26
40	0.21	0.33	57.21
50	0.21	0.35	57.07
60	0.21	0.36	55.80
70	0.20	0.43	53.22
80	0.19	0.43	50.35
90	0.18	0.55	49.40
100	0.17	0.62	46.37

#### 4.2 Case 2: Transmit Time

Transmit time or transmission time is the time starting from transmitting first bit of message till the last bit of message leaves the transmission link.

$$\text{Transmit time} = \text{File Size (Mbyte)} / \text{Bandwidth (Mbit/sec)}$$

In second case, we consider link's 'transmit time' as a performance parameter. The size of a file is 10 MB. Using above formula, we find the average transmit time for each server and assign the weights according to least transmit time. Table 4.7 shows transmit time of each server.

Table 4.7: Transmit Time ( $\mu\text{s}$ )

Servers	Transmit Time ( $\mu\text{s}$ )
Server 1	1057.95
Server 2	1033.60
Server 3	1121.465

Server 2 has least transmit time; hence we assign highest weight to it, consequently more requests are forwarded to server 2. Weight assigned to server 1 is higher than server 3, but less than server 2.

#### 4.2.1 WRR (r=1):

In table 4.8, we find the required results by applying weighted round robin for 'r' equal to one with different number of users.

Table 4.8: Weighted Round Robin for  $r = 1$

No. of Users	Throughput (Mbit/sec)	Response time (sec)	Transaction rate (Trans/sec)
10	0.27	0.13	72.64
20	0.25	0.24	68.26
30	0.25	0.31	66.06
40	0.23	0.46	62.65
50	0.21	0.57	57.35
60	0.21	0.63	56.86
70	0.20	0.63	54.73
80	0.20	0.63	53.05
90	0.19	0.74	50.74
100	0.18	0.82	47.26

#### 4.2.2 WRR (r=5):

Table 4.9 shows the results for increased data traffic by setting number of concurrent users (-c) from 10 to 100 with -r equal to '5'.

Table 4.9: Weighted Round robin for r = 5

No. of Users	Throughput (Mbit/sec)	Response time (secs)	Transaction rate (trans/sec)
10	0.27	0.15	71.24
20	0.26	0.28	70.47
30	0.24	0.37	64.25
40	0.24	0.45	64.07
50	0.23	0.51	61.84
60	0.22	0.56	59.47
70	0.22	0.59	58.12
80	0.21	0.61	57.76
90	0.21	0.66	56.08
100	0.17	0.70	46.62

### 4.3 Case 3: Transfer Time

A time interval between starting and completion of transfer of a file is called 'Transfer time'. It also includes the additional time spent on setting or requesting transfer.

$$\text{Transfer time} = \text{Round trip time (ms)} + 1 / (\text{bandwidth (Mbps)} \times \text{transfer size (MB)})$$

In third case, we consider the link 'transfer time' as a performance parameter. The size of a file is 1MB. Using the above formula, we find the average transfer time to each server and assign weights according to least transfer time. Table 4.10 shows transfer time of each server.

Table 4.10: Transfer Time

Servers	Transfer Time (ms)
Server 1	4.54017
Server 2	5.08114
Server 3	4.82001



Server 1 has least transmit time, so highest weight is assigned to it resulting in most requests forwarded to it. Assign weight to server 3 higher than server 2, but less than server 1.

### 4.3.1 Weighted Round Robin (r=1):

In table 4.11, we need to find transfer time by applying weighted round robin for 'r' equal to one with different number of users. First of all, we find the average transfer time of each server. The required quantities to calculate transfer time are round trip time, bandwidth of the link and size of file. After getting results, we assign weight to each server w.r.t the least transfer time and run the load balancing application.

Table 4.11: Weighted Round Robin for r = 1

Users	Response time (secs)	Transaction rate (trans/sec)	Throughput (Mbit/sec)
10	0.13	70.18	0.26
20	0.14	69.43	0.26
30	0.31	65.04	0.24
40	0.48	65.18	0.24
50	0.53	58.09	0.22
60	0.62	43.43	0.16
70	0.66	42.74	0.16
80	0.66	39.49	0.15
90	0.68	39.06	0.15
100	0.75	35.54	0.13

### 4.3.2 Weighted Round Robin (r=5):

Table 4.12 shows the results for increased data traffic by setting number of concurrent users (-c) from 10 to 100 with -r equals to '5'.

Table 4.12: Weighted Round Robin for  $r = 5$

Users	Response time (secs)	Transaction rate (trans/sec)	Throughput (Mbit/sec)
10	0.13	72.62	0.27
20	0.22	69.49	0.26
30	0.32	64.31	0.24
40	0.33	58.07	0.22
50	0.53	55.61	0.21
60	0.66	49.80	0.19
70	0.71	46.54	0.17
80	0.84	46.64	0.17
90	0.81	39.76	0.15
100	0.87	35.45	0.13

In this chapter, we are done with the different considered scenarios and testbed results after implementing Round Robin and proposed Weighted Round Robin algorithm. Now in Chapter 5, we analyze and discuss our results in terms of response time, transaction rate and throughput.

# Chapter 5

## Result Analysis and Discussion

In this chapter, results of proposed work have been discussed. The performance parameters like response time, transaction rate and throughput of both round robin and weighted round robin are compared. As it can be visualized from the further presented calculations and graphical previews it can be deduced that weighted round robin gives better results.

### 5.1 Case 1: Bandwidth Delay Product ( $r=1$ )

As discussed earlier, we used bandwidth delay product for WRR. Now testing our policy for  $r=1$ .

#### 5.1.1 Response Time:

Response time is a time taken by server to respond to each user's request. Fig. 5.1 shows response time of both algorithms. The response time for RR algorithm ranges from 0.22-2.45 sec. Whereas, response time for WRR ranges from 0.11 to 0.79 sec. Proposed WRR method achieves lesser response time than RR.

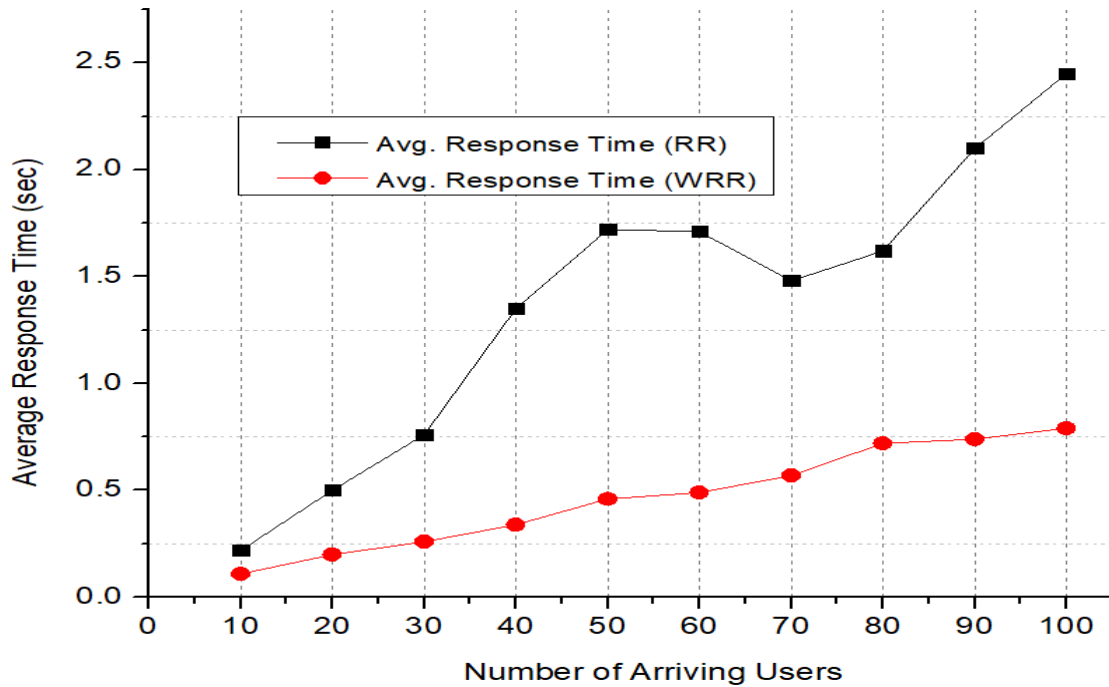


Figure 5.1: Response Time in relation to number of Arriving clients

### 5.1.2 Transaction Rate:

Transaction rate is the number of transactions per second that a server has to handle. Transaction rate of a server should be high so it can handle maximum requests. The transaction rate for RR ranges from 8.7trans/sec to 41.01trans/sec. whereas, transaction rate for WRR ranges from 56.24 to 88.21 trans/sec. In our working, we have achieved higher transactions than Round Robin. Fig. 5.2 shows the transaction rate graph.

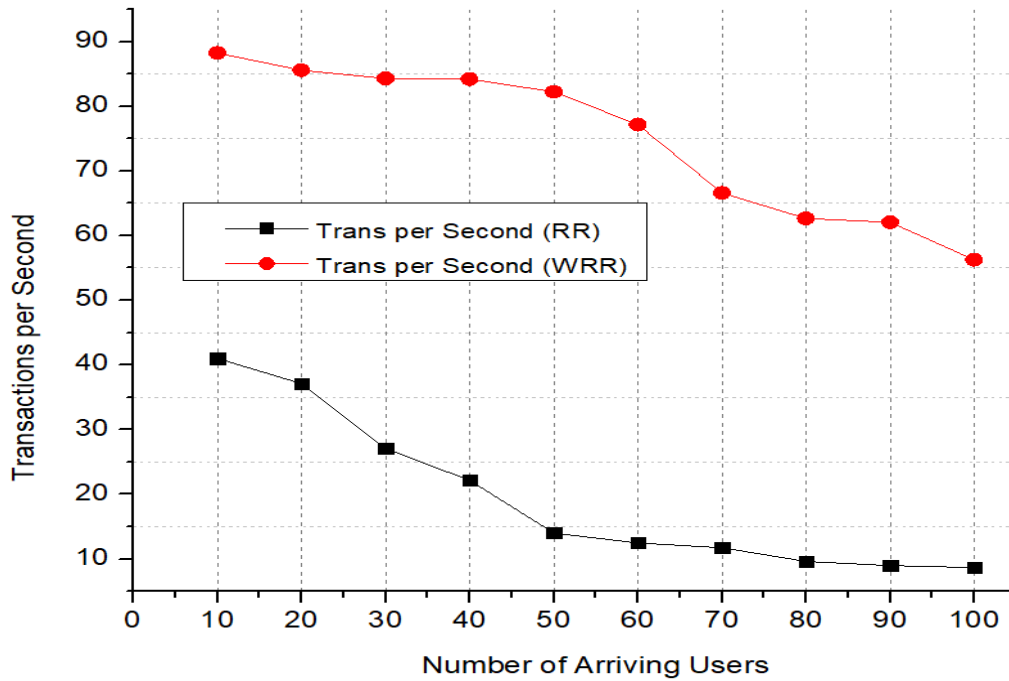


Figure 5.2: Transaction rate in relation to number of arriving Clients

### 5.1.3 Throughput:

On basis of least bandwidth delay product, we achieve high throughput rate for WRR as compared to RR. Fig. 5.3 shows the throughput graph. The throughput rate for RR ranges from 0.03 Mbit/sec to 0.13 Mbit/sec. whereas, the throughput rate for WRR ranges from 0.21 to 0.33 Mbit/sec.

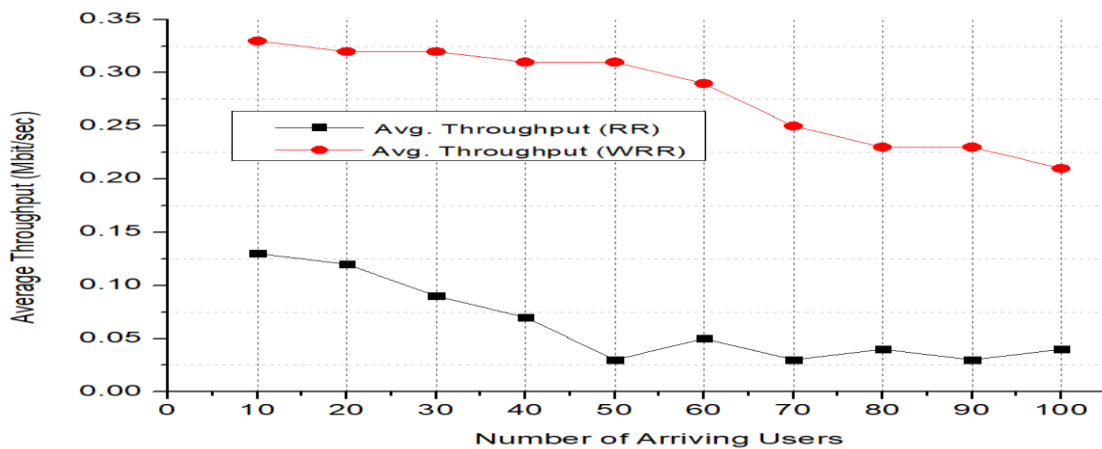


Figure 5.3: Throughput in relation to number of Arriving Clients

## 5.2 Bandwidth Delay Product ( $r = 5$ ):

### 5.2.1 Response Time:

Fig. 5.4 shows the response time when increasing the number of clients. By increasing number of clients, by value of 'r' equals to five, the average load increases also. For RR, the response time ranges from 0.22 to 1.11 sec. The response time for WRR is from 0.15 to 0.62 sec. We achieve better response time for increasing arriving clients.

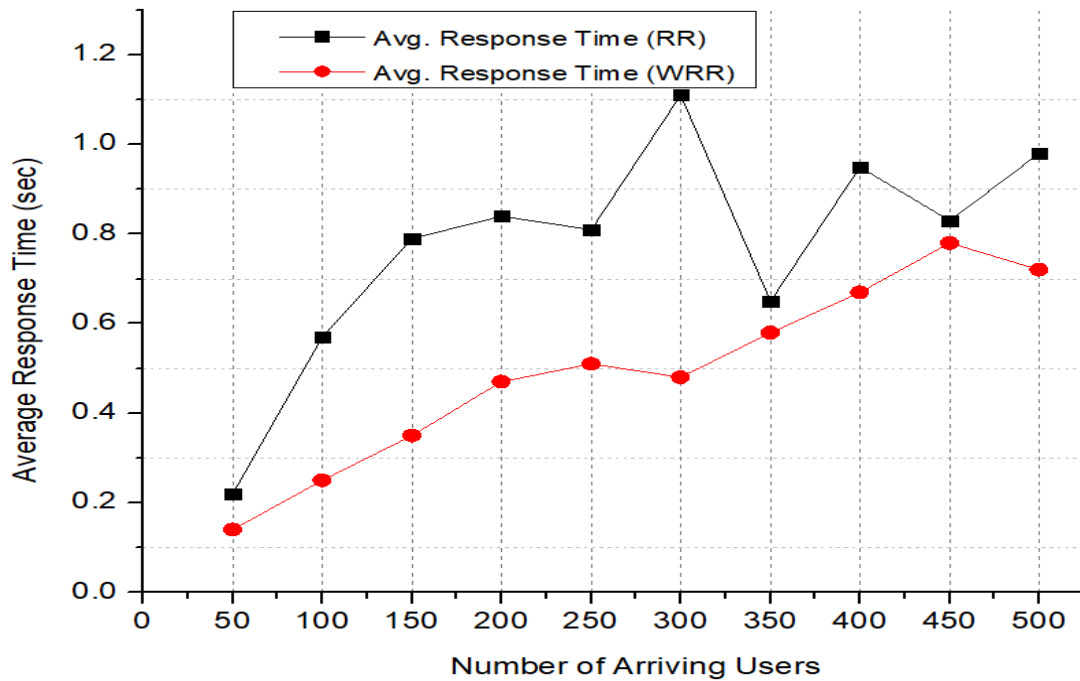


Figure 5.4: Response time in relation to number of Arriving Clients

### 5.2.2 Transaction Rate:

By increasing load, we find here number of transaction per second for both algorithms.

Fig. 5.5 shows the transaction rate graph. The transaction rate for RR ranges from 10.18 trans/sec to 43.17 trans/sec. whereas, the transaction rate for WRR ranges from 46.37 trans/sec to 63.89 trans/sec. In our work, we achieve high transactions rate than RR.

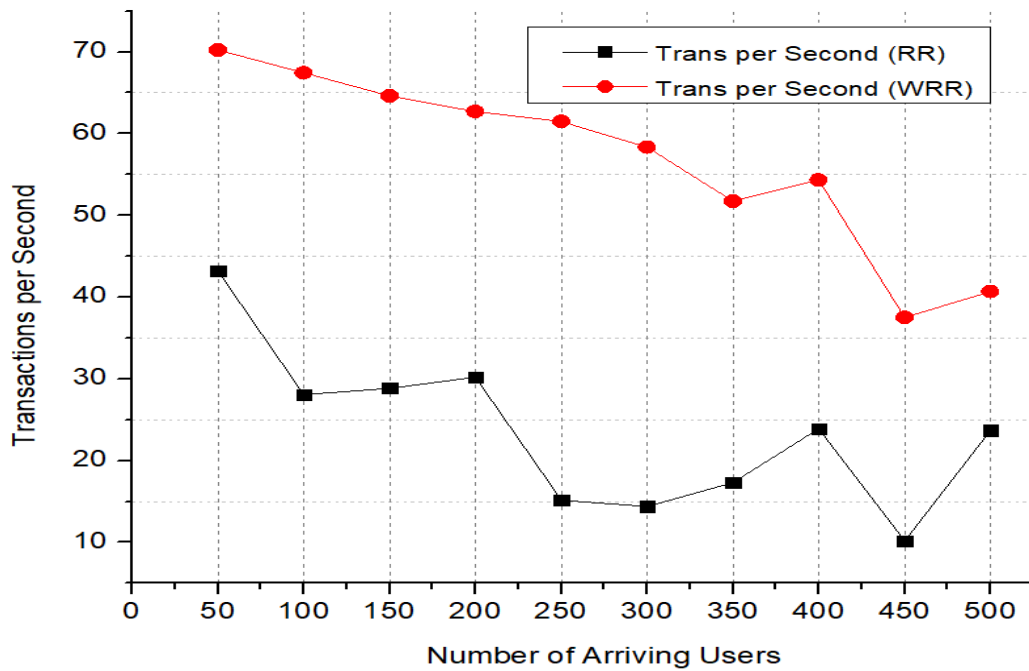


Figure 5.5: Transaction Rate in relation to number of Arriving Clients

### 5.2.3 Throughput:

On basis of least bandwidth delay product, we achieve higher throughput rate for Weighted Round Robin (WRR) as compared to Round Robin (RR). Fig. 5.6 shows the throughput graph of increased average load. The throughput rate for RR ranges from 0.03Mbit/sec to 0.13Mbit/sec. whereas, the throughput rate for Weighted Round Robin (WRR) ranges from 0.17Mbit/sec to 0.24Mbit/sec.

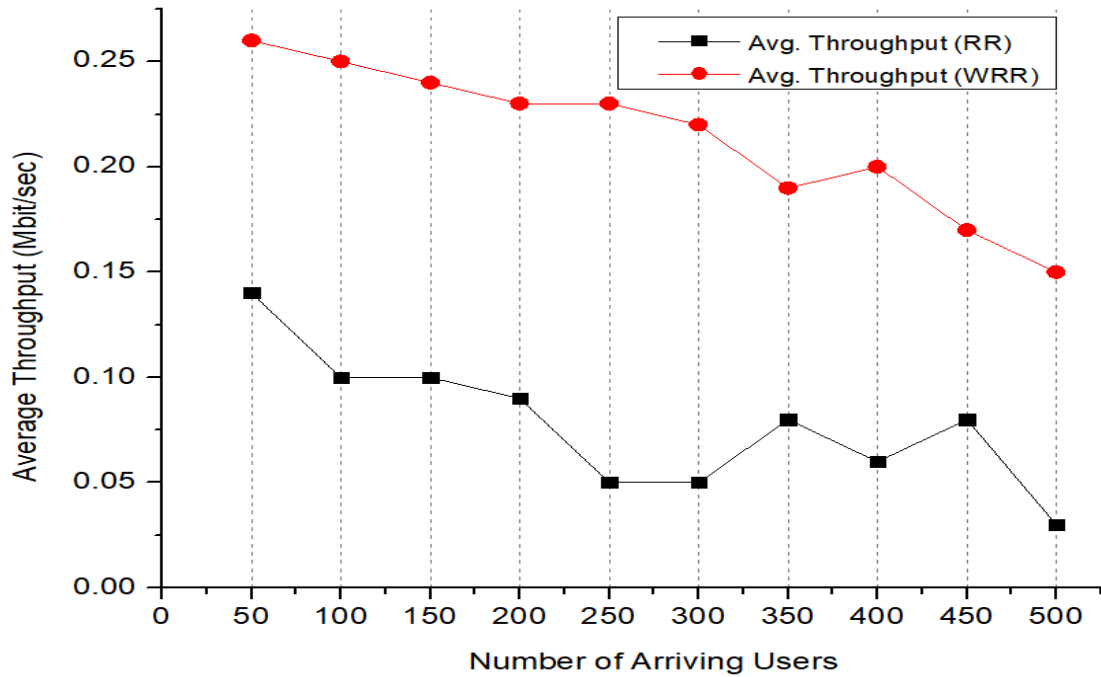


Figure 5.6: Throughput in relation to number of Arriving Client

### 5.3 Case 2 Transmit Time: ( $r=1$ )

As described earlier, transmit time is used as second parameter for our observations for WRR. We test our policy for  $r=1$ . Fig. 5.7 shows the response time of both algorithms.

#### 5.3.1 Response time:

The response time for Round Robin algorithm ranges from 0.22ms to 2.45ms. Whereas, the response time for Weighted Round R ranges from 0.13 to 0.82 sec. Proposed WRR achieves response time that is lesser than RR.



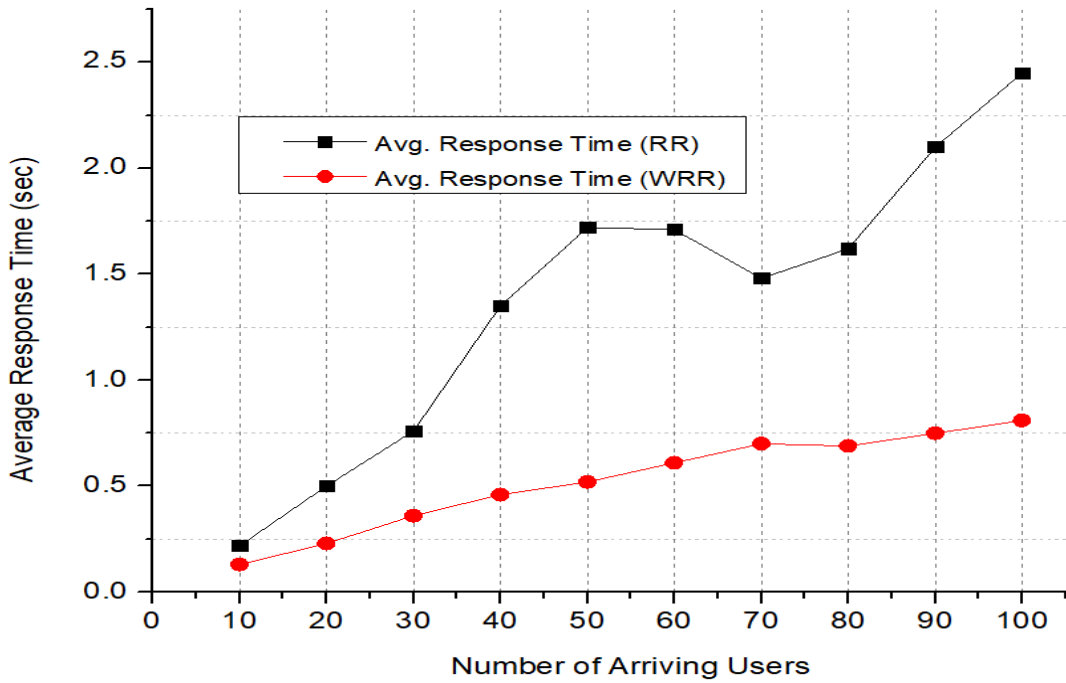


Figure 5.7: Response time in relation to number of Arriving Clients

### 5.3.2 Transaction rate:

Fig. 5.8 shows the transaction rate on basis of least transmit time. The transaction rate for WRR ranges from 47.26 trans/sec to 72.64 trans/sec. In our work, we achieve high transactions rate than RR.

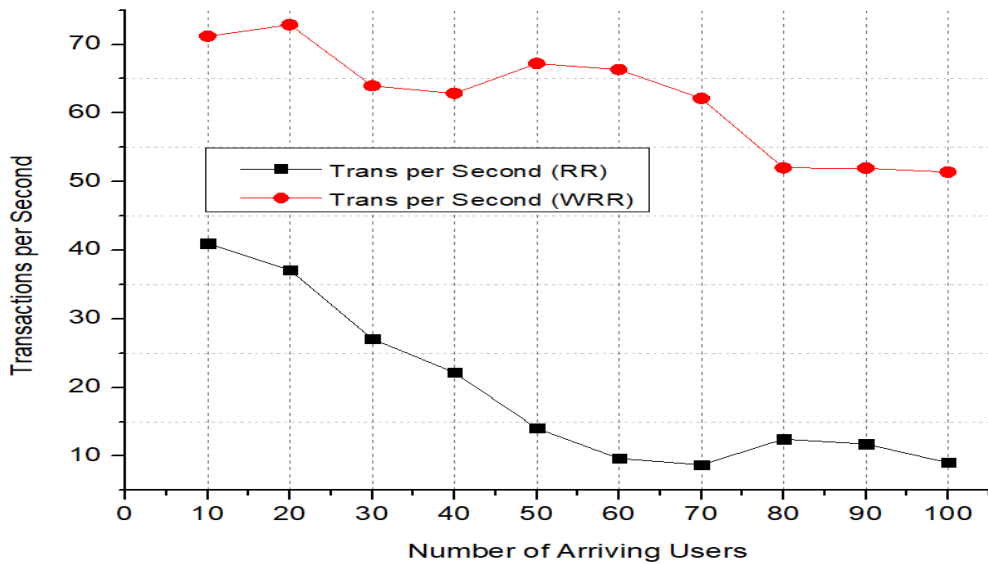


Figure 5.8: Transaction rate in relation to number of Arriving Clients

### 5.3.3 Throughput:

Fig. 5.9 shows better throughput rate for WRR than RR for 'r' equals to one. WRR throughput ranges from 0.18 to 0.27Mbit/sec, showing better throughput rate than RR.

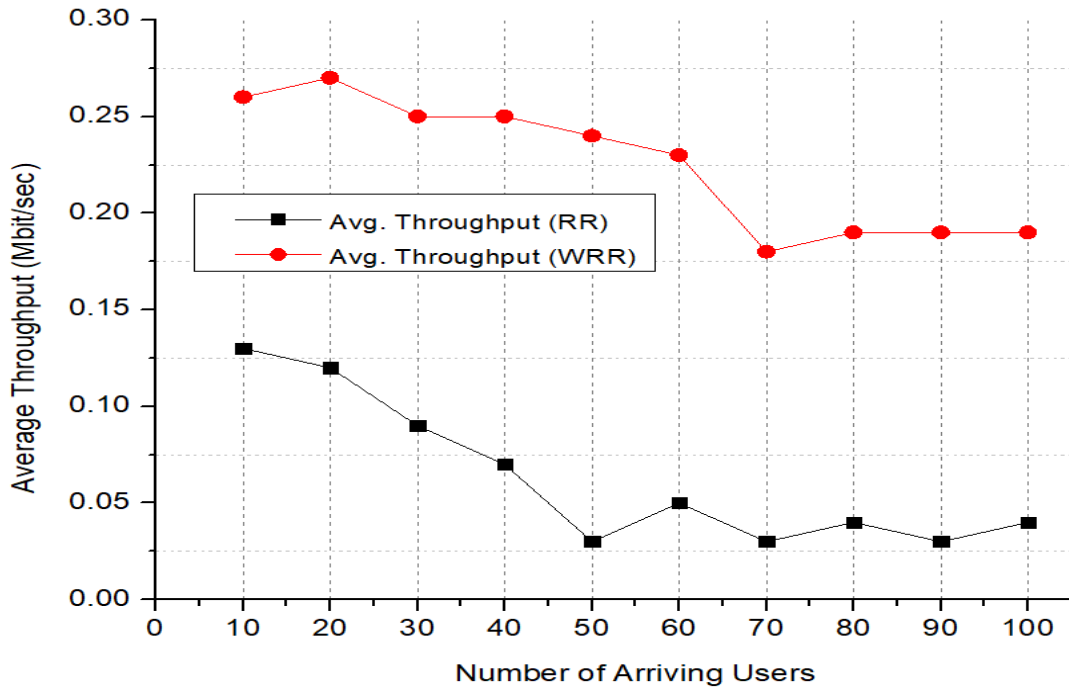


Figure 5.9: Throughput in relation to number of Arriving Clients

### 5.4 Transmit time ( $r = 5$ ):

Based on least transmit time; we increase the average load or traffic by value of five, while comparing these results with that of RR.

#### 5.4.1 Response Time:

Fig. 5.10 shows the response time by increasing average load. The response time for RR algorithm ranges from 0.22ms to 1.11m. Whereas, the response time for WRR ranges from 0.15 to 0.70 sec. Proposed WRR achieves less response time than RR.

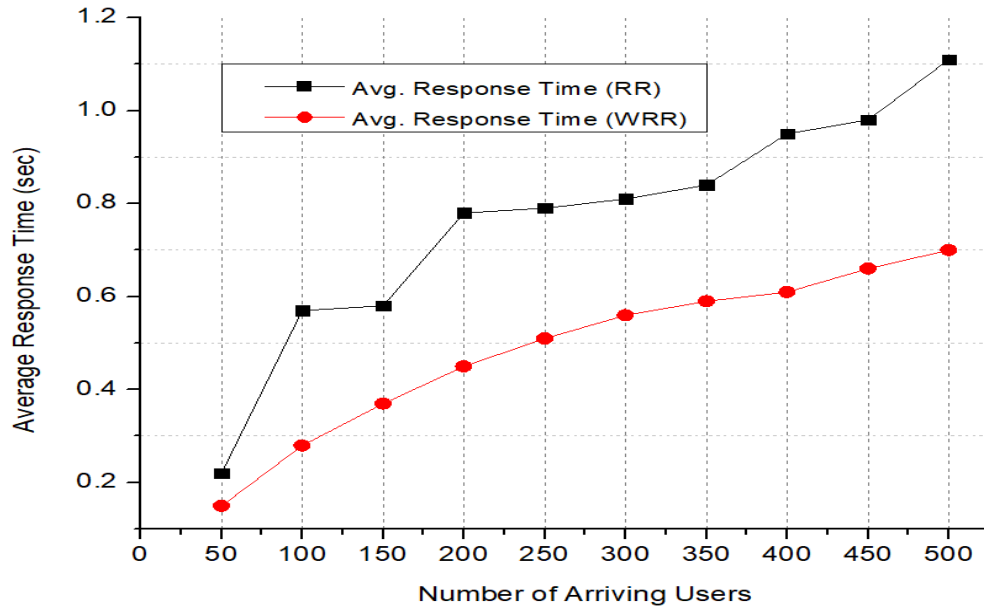


Figure 5.10: Response time in relation to number of Arriving Clients

### 5.4.2 Transaction Rate:

Fig. 5.11 shows the transaction rate on basis of least transmit time. The transaction rate for RR ranges from 10.16trans/sec to 43.17trans/sec. The transaction rate for WRR ranges from 46.62 to 71.24 trans/sec. In our proposed method, higher transaction rates are achieved than RR.

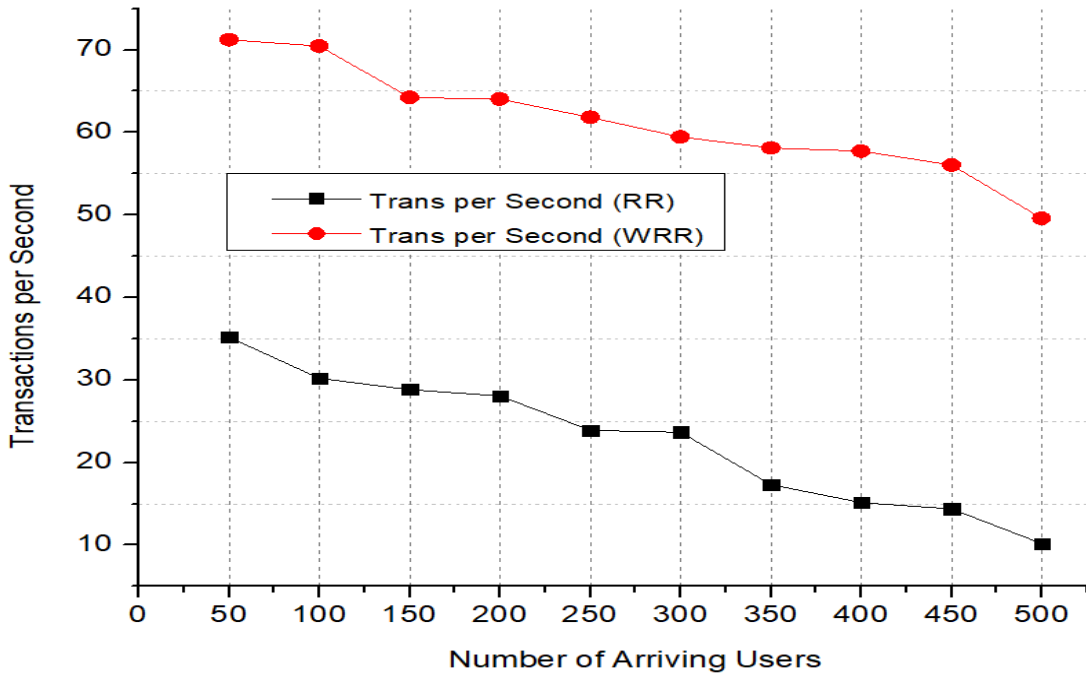


Figure 5.11: Transaction rate in relation to number of Arriving Clients

### 5.4.3 Throughput:

Fig. 5.12 shows that WRR has better throughput rate than RR for 'r' equal to five. The RR throughput ranges from 0.03 to 0.14Mbit/sec, Throughput for WRR ranges from 0.17 to 0.27Mbit/sec, which clearly shows WRR throughput is better throughput rate than RR.

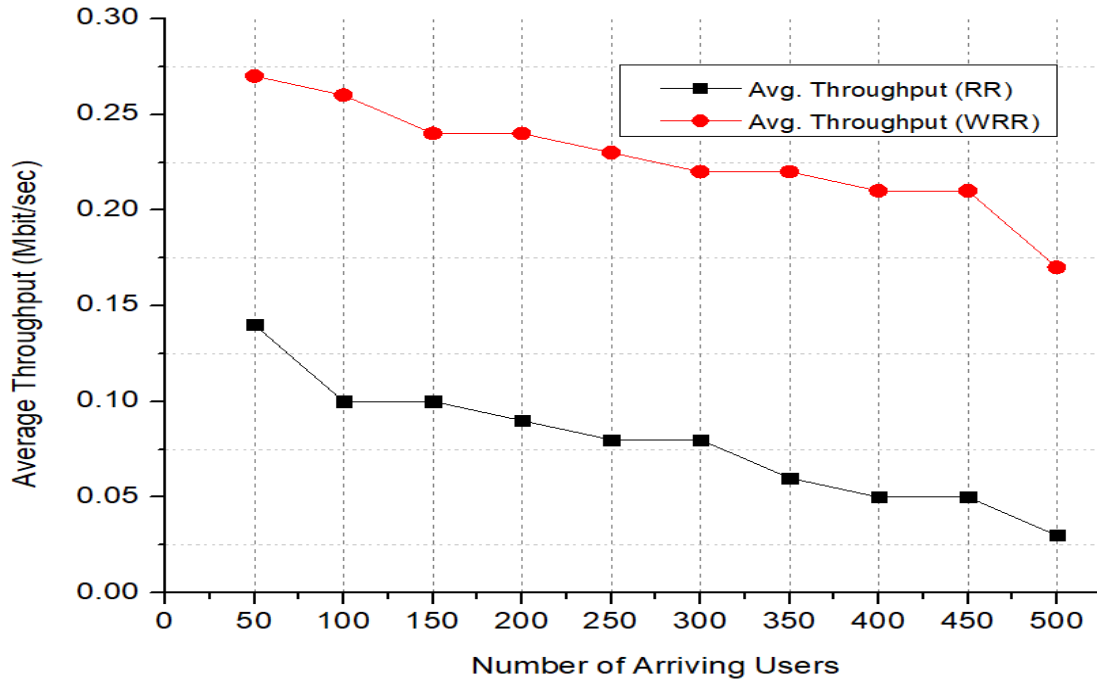


Figure 5.12: Throughput in relation to number of Arriving Clients

## 5.5 Case 3 Transfer Time: ( $r=1$ )

As discussed in chapter 4, we used transfer time as the third performance parameter for WRR. Now testing it as per our procedure for  $r=1$ .

### 5.5.1 Response Time:

Fig. 5.13 shows the response time comparison of both algorithms. The response time for RR algorithm ranges from 0.22ms to 2.45ms. Whereas, the response time for WRR ranges from 0.13 to 0.75 sec. Using proposed WRR methodology lesser response time than RR can be achieved.

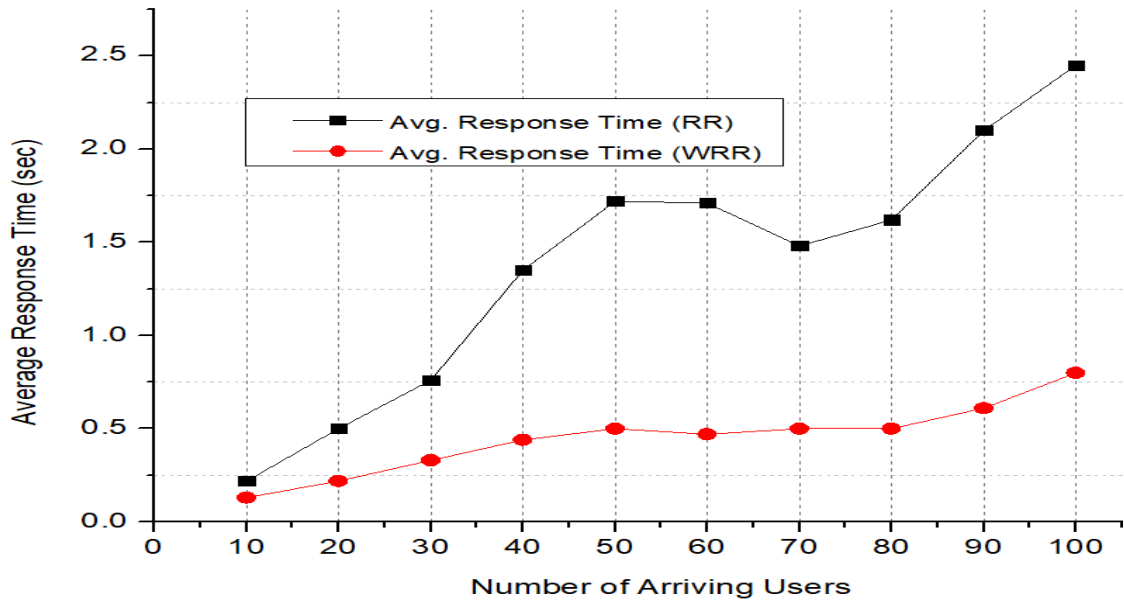


Figure 5.13: Response time in relation to number of Arriving Clients

### 5.5.2 Transaction Rate:

Fig. 5.14 shows the transaction rate on basis of least transfer time. In WRR, transaction rate for each client is higher than RR. The transaction rate for WRR ranges from 35.54 to 70.18 trans/sec, which is a much higher achieved transaction rate than simple RR algorithm.

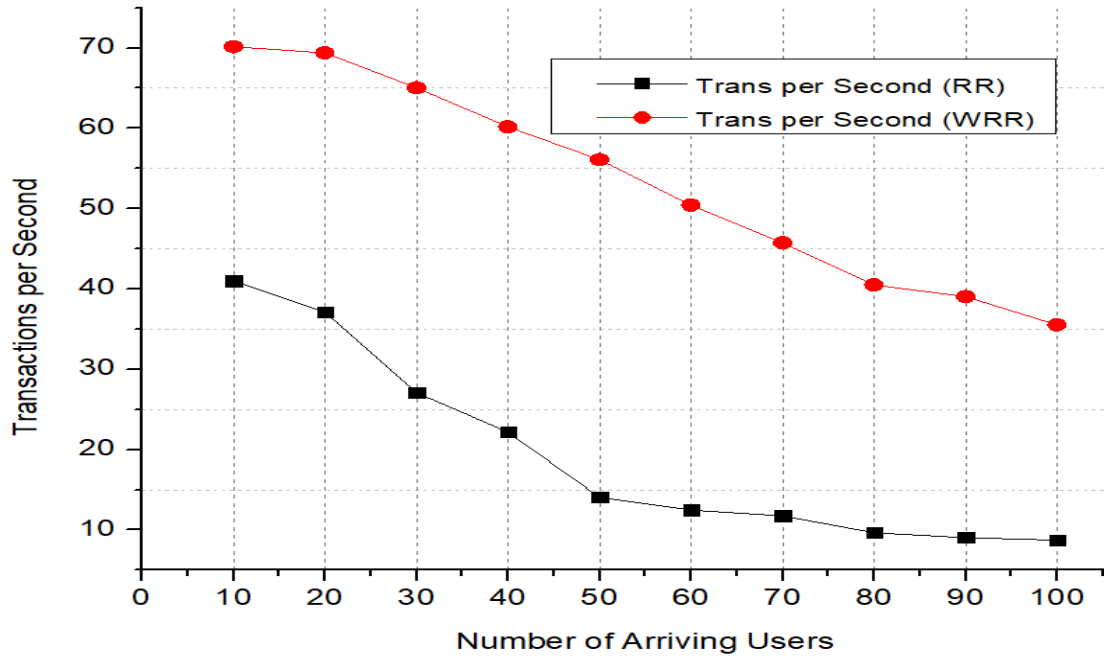


Figure 5.14: Transaction rate in relation to number of Arriving Clients

### 5.5.3 Throughput:

Fig. 5.15 shows that better throughput rate is being achieved for WRR than RR for unchanged value of 'r' i.e. equal to one. The RR throughput ranges from 0.03 to 0.14Mbit/sec. WRR throughput ranges from 0.13 to 0.26 Mbit/sec, hence it can be clearly seen that WRR has better throughput rate than RR.

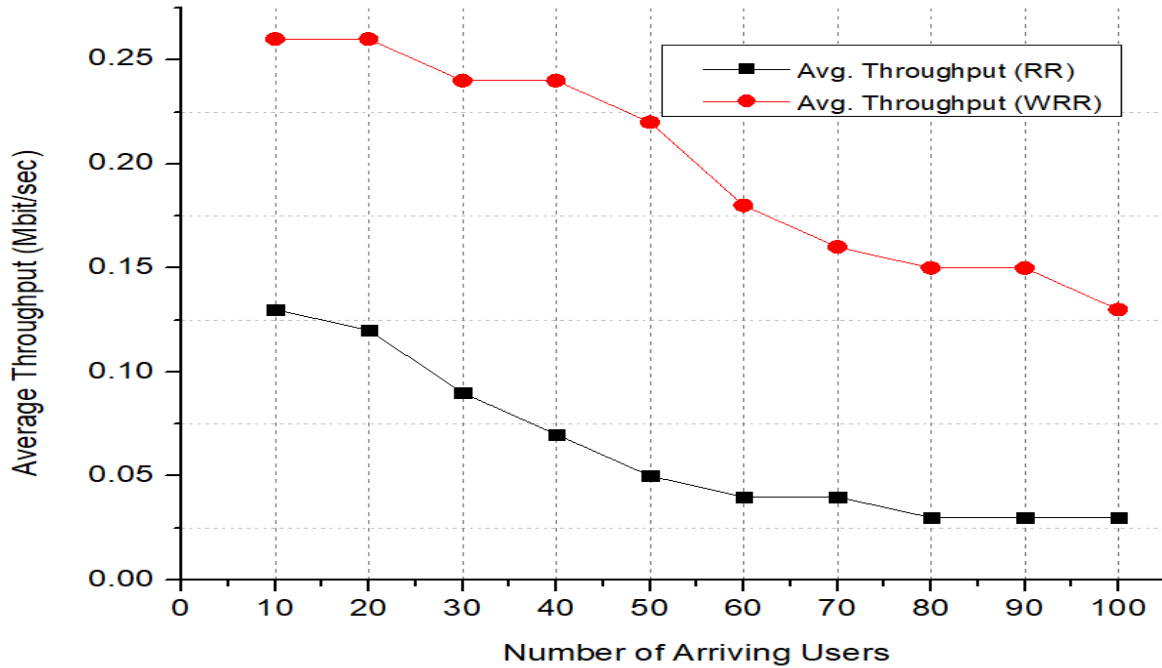


Figure 5.15: Throughput in relation to number of Arriving Clients

## 5.6 Case 3 Transfer Time: (r=5)

Now using transfer time as a performance parameter for WRR, testing it while keeping r=5.

### 5.6.1 Response Time:

Fig. 5.16 shows the response time comparison of both algorithms. The response time for RR algorithm ranges from 0.22ms to 1.11ms. whereas, response time for WRR ranges from 0.13 to 0.87 sec. Proposed WRR achieves response time less than RR.



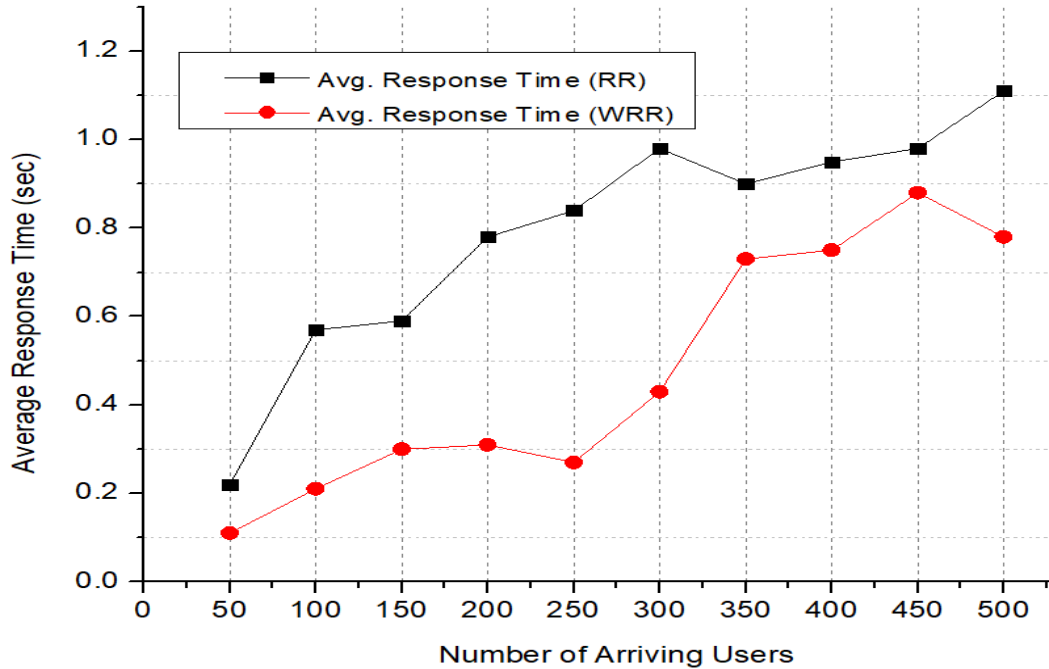


Figure 5.16: Response time in relation to number of Arriving Clients

### 5.6.2 Transaction Rate:

Fig5.17 shows transaction rate based on least transfer time. The transaction rate for WRR ranges from 35.45 to 72.62 trans/sec, achieving higher transactions rate than RR.

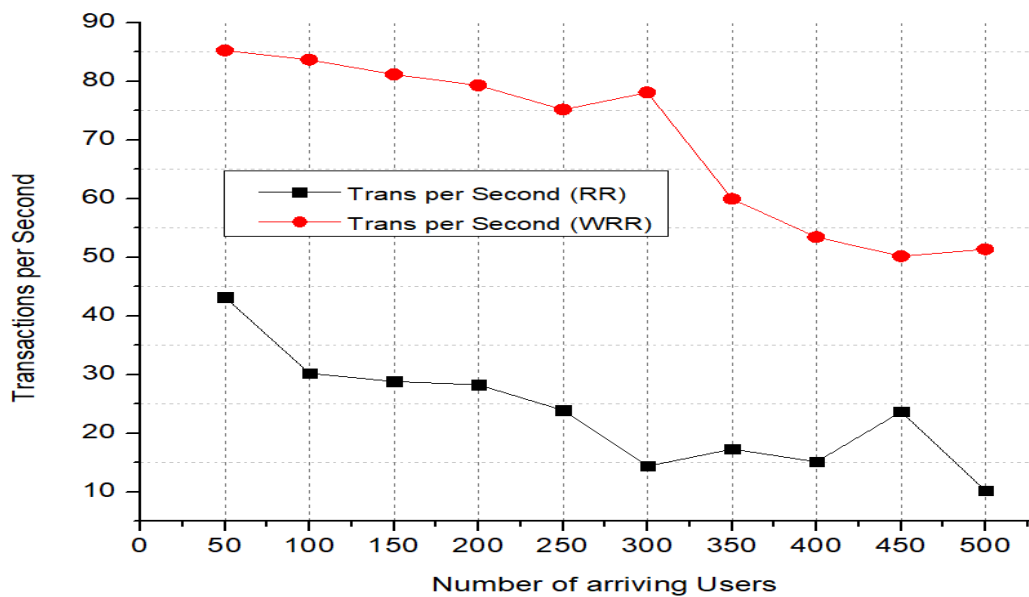


Figure 5.17: Transaction rate in relation to number of Arriving Clients

### 5.6.3 Throughput:

Fig. 5.18 shows that throughput rate is better for WRR than RR taking 'r' equal to five. The RR throughput ranges from 0.03 to 0.14 Mbit/sec, however WRR throughput ranges from 0.13 to 0.27 Mbit/sec, depicting better throughput rate than RR.

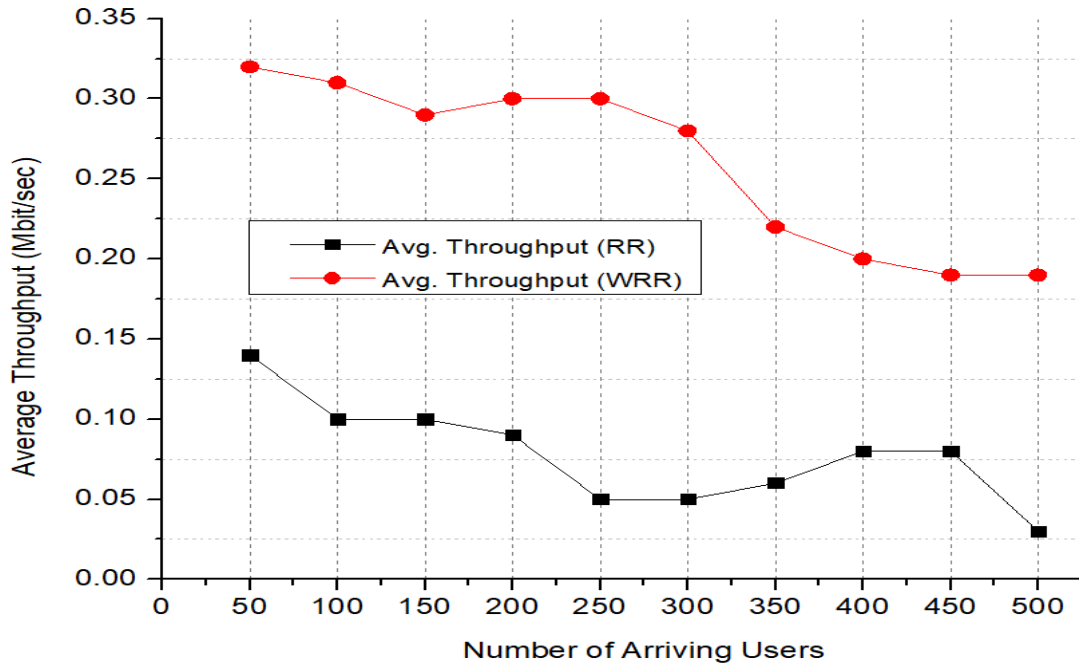


Figure 5.18: Throughput in relation to number of Arriving Clients

In this chapter, we have dealt with the discussion and results of Round Robin and proposed Weighted Round Robin method in the form of graphical representation. From all the results, better performance results are attained using proposed Weighted Round Robin as compared to Round Robin. We have derived these results by determining three network parameters that were bandwidth delay product, transmission time and transfer time. From all above outcomes, we conclude our work in chapter 6.

Table 5.1: Result analysis using Round Robin and Weighted Round Robin

Parameters	Round Robin		Weighted Round Robin		Weighted Round Robin		Weighted Round Robin	
			Bandwidth Delay		Transmit Time		Transfer Time	
	r = 1	r = 5	r = 1	r = 5	r = 1	r = 5	r = 1	r = 5
Response Time (Secs)	0.22	0.22	0.11	0.15	0.13	0.15	0.13	0.13
	2.45	1.11	0.79	0.62	0.82	0.70	0.75	0.87
Transaction Rate (Trans/sec)	8.70	10.18	56.24	46.37	47.26	46.62	35.54	35.45
	41.01	43.17	88.21	63.89	72.64	71.24	70.18	72.62
Throughput (Mbit/sec)	0.03	0.03	0.21	0.17	0.18	0.17	0.13	0.13
	0.13	0.14	0.33	0.24	0.27	0.27	0.26	0.27

Table 5.1 shows the analysis between Round Robin and Weighted Round Robin results. However, Weighted Round Robin has better results for all parameters in general than Round Robin algorithm

# Chapter 6

## **Conclusion and Future Work**

In chapter 5, we had been comparing the results of both load-balancing techniques. Both Round Robin and Weighted Round Robin algorithms are simulated in a SDN environment. The acquired results are compared in terms of response time (sec), transaction rate (trans/sec) and throughput (Mbit/sec). Comparing results shows that our proposed parameters for Weighted Round Robin provides better results than Round Robin algorithm.

In future work, evaluation of the above-mentioned algorithms can be done in any other SDN based controller like Ryu, floodlight etc. In our work, we had used single controller but for future work, multi controllers can be used and implemented. Different network topologies like fat tree, diamond, mesh or any customized topology consisting of different number of switches, servers and clients can be used to test the application as well.

## References:

1. Braun, Wolfgang, and Michael Menth. "Software-defined networking using OpenFlow: Protocols, applications and architectural design choices." *Future Internet*, pp. 302-336, 2014.
2. A. Blenk, A. Basta, M. Reisslein and W. Kellerer, "Survey on Network Virtualization Hypervisors for Software Defined Networking," in *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 655-685, Firstquarter 2016.
3. Jammal, Manar, et al. "Software defined networking: State of the art and research challenges." *Computer Networks*, vol. 72, pp. 74-98, 2014.
4. Z. Qin, G. Denker, C. Giannelli, P. Bellavista and N. Venkatasubramanian, "A Software Defined Networking architecture for the Internet-of-Things," *2014 IEEE Network Operations and Management Symposium (NOMS)*, Krakow, pp. 1-9, 2014.
5. H. Kim, J. Kim and Y. Ko, "Developing a cost-effective OpenFlow testbed for small-scale Software Defined Networking," *16th International Conference on Advanced Communication Technology*, Pyeongchang, pp. 758-761, 2014.
6. A. Lara, A. Kolasani and B. Ramamurthy, "Simplifying network management using Software Defined Networking and OpenFlow," *2012 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Bangalore, pp. 24-29, 2012.
7. T. Luo, H. Tan and T. Q. S. Quek, "Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks," in *IEEE Communications Letters*, vol. 16, no. 11, pp. 1896-1899, November 2012.
8. S. Kaur, K. Kumar, J. Singh and Navtej Singh Ghumman, "Round-robin based load balancing in Software Defined Networking," *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, pp. 2136-2139, 2015.
9. H. Kim and N. Feamster, "Improving network management with software defined networking," in *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114-119, February 2013.

10. W. Xia, Y. Wen, C. H. Foh, D. Niyato and H. Xie, "A Survey on Software-Defined Networking," in *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27-51, Firstquarter 2015.
11. B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," in *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617-1634, 2014.
12. S. H. Yeganeh, A. Tootoonchian and Y. Ganjali, "On scalability of software-defined networking," in *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136-141, February 2013.
13. K. Bao, J. D. Matyjas, F. Hu and S. Kumar, "Intelligent Software-Defined Mesh Networks With Link-Failure Adaptive Traffic Balancing," in *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 2, pp. 266-276, June 2018.
14. S. Ejaz, Z. Iqbal, P. Azmat Shah, B. H. Bukhari, A. Ali and F. Aadil, "Traffic Load Balancing Using Software Defined Networking (SDN) Controller as Virtualized Network Function", *IEEE Access*, Vol. 6, pp. 46646-46658, 2019.
15. G. Li, X. Wang and Z. Zhang, "SDN Based Load Balancing Scheme for Multi-Controller Deployment", *IEEE Access*, Vol. 7, pp. 39612-39622, 2019.
16. V. Varadharajan, K. Karmakar, U. Tupakula and M. Hitchens, "A Policy-Based Security Architecture for Software-Defined Networks," in *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 897-912, April 2019.
17. D. Kreutz, F.M.V. Ramos, P.E. Verissimo, C.E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey", *Proceedings of IEEE*, Vol. 103, pp. 14-76, Jan 2015.
18. 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, Dec 2018.
19. R. Trestian, K. Katrinis and G. Muntean, "OFLoad: An OpenFlow Based Dynamic Load Balancing Strategy for Datacenter Networks", *IEEE Transactions on Network and Service Management*, Vol. 14, no. 4, pp. 792-803, Dec 2017.

20. W. Xia, Y. Wen, C. H. Foh, D. Niyato and H. Xie, "A Survey on Software-Defined Networking," in *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27-51, Firstquarter2015.
21. Shalimov, Alexander, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky. "Advanced study of SDN/OpenFlow controllers." In *Proceedings of the 9th central & eastern european software engineering conference in russia*, pp. 1-6. 2013.
22. Song, Haoyu. "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane." In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 127-132. 2013.
23. S. Kim, H. Choi, P. Park, S. Min and Y. Han, "OpenFlow-based Proxy mobile IPv6 over software defined network (SDN)," *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, Las Vegas, NV, pp. 119-125, 2014.
24. S. Wang, "Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet," *2014 IEEE Symposium on Computers and Communications (ISCC)*, Funchal, pp. 16, 2014.
25. J. Medved, R. Varga, A. Tkacik and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, Sydney, NSW, pp. 1-6, 2014.
26. S. Tomovic, N. Prasad and I. Radusinovic, "SDN control framework for QoS provisioning," *2014 22nd Telecommunications Forum Telfor (TELFOR)*, Belgrade, pp. 111-114, 2014.
27. C. Donato, P. Serrano, A. de la Oliva, A. Banchs and C. J. Bernardos, "An openflow architecture for energy-aware traffic engineering in mobile networks," in *IEEE Network*, vol. 29, no. 4, pp. 54-60, July-August 2015.
28. J. Mambretti, J. Chen and F. Yeh, "Software-Defined Network Exchanges (SDXs): Architecture, services, capabilities, and foundation technologies," *2014 26th International Teletraffic Congress (ITC)*, Karlskrona, 2014, pp. 1-6.

29. M. Hayes, B. Ng, A. Pekar and W. K. G. Seah, "Scalable Architecture for SDN Traffic Classification," in *IEEE Systems Journal*, vol. 12, no. 4, pp. 3203-3214, Dec. 2018.
30. Bholebawa, Idris Zoher, Rakesh Kumar Jha, and Upena D. Dalal. "Performance analysis of proposed OpenFlow-based network architecture using mininet." *Wireless Personal Communications* 86, no. 2 (2016): 943-958.
31. A. Blenk, A. Basta, J. Zerwas, M. Reisslein and W. Kellerer, "Control Plane Latency With SDN Network Hypervisors: The Cost of Virtualization," in *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 366-380, Sept. 2016.
32. A. A. Neghabi, N. Jafari Navimipour, M. Hosseinzadeh and A. Rezaee. "Load Balancing Mechanisms in Software Defined Networks: A Systematic and Comprehensive Review of Literature", *IEEE Access*, Vol. 6, pp. 14159-14178, 2018.
33. S. Aymaz, T. Cavdar, S. Aymaz and E. Ozturk, "An Analysis of Load Balancing Strategies with Wireshark in Software Defined Networks", International Conference on Artificial Intelligence and Data Processing (IDAP), Malatya, Turkey, pp. 1-5, 2018.
34. Cortes, Antonio, "Simulation of Software Defined Networks with Open Network Operating System and Mininet", *International Journal of Computer Science & Information Technology (IJCSIT)*, Vol. 10, no. 5, Oct 2018.
35. M. Alsaeedi, M. M. Mohamad and A. A. Al-Roubaiey, "Toward Adaptive and Scalable OpenFlow-SDN Flow Control: A Survey," in *IEEE Access*, vol. 7, pp. 107346-107379, 2019.
36. Muragaa, W. Seman and K. Marhusin, "A POX Controller Module to Prepare a List of Flow Header Information Extracted from SDN Traffic." 2017.
37. T. Huang, F. R. Yu, C. Zhang, J. Liu, J. Zhang and Y. Liu, "A Survey on Large-Scale Software Defined Networking (SDN) Testbeds: Approaches and Challenges," in *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 891-917, Second quarter 2017..
38. Yuba Stanford, [Online]. Available: <http://yuba.stanford.edu/cs244wiki/index.php/Overview>.



39. ONOS Project (ONOS), [online]. Available: <https://wiki.onosproject.org>.
40. Open Networking Foundation (ONF), [Online]. Available: <https://www.opennetworking.org/>
41. J. Tourrilhes, P. Sharma, S. Banerjee and J. Pettit, "SDN and OpenFlow Evolution: A Standards Perspective", *Computer Networks*, Vol. 47, no. 11, pp. 22-29, Nov 2014.
42. M. Aslan and A. Matrawy, "On the Impact of Network State Collection on the Performance of SDN Applications," in *IEEE Communications Letters*, vol. 20, no. 1, pp.5-8, Jan.2016.
43. S. Khan, A. Gani, A. W. Abdul Wahab, M. Guizani and M. K. Khan, "Topology Discovery in Software Defined Networks: Threats, Taxonomy, and State-of-the-Art," in *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 303-324, Firstquarter 2017.
44. M. Paliwal, D. shrimankar and O. Tembhurne, "Controllers in SDN: A Review Report", *IEEE Access*, Vol. 6, pp. 36256-36270, 2018.
45. C. Hao, Chang and Y. D. Lin, "OpenFlow Version Roadmap", 2015.
46. Y. Yu *et al.*, "Fault Management in Software-Defined Networking: A Survey," in *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 349-392, Firstquarter 2019.
47. I.F. Akyildiz, A. Lee, P. Wang, M. Luo, W. Chou, "A roadmap for traffic engineering in SDN OpenFlow Networks", *Computer Networks*, Vol. 71, pp. 1-30, Oct 2014.
48. <https://www.informit.com/articles/article.aspx?p=2451956&seqNum=3>.
49. S. Wison and D. akshmi, "Dynamic Server Load Balancing algorithm", *International Journal of Communication Systems*, 2018.
50. Y. Wang and s. You, "An Efficient Route Management Framework for Load Balance and Overhead Reduction in SDN Based Datacenter Networks", *IEEE Transactions on Network and Service Management*, Vol. 15, no. 4, pp. 1422-1434, Dec 2018.
51. Saisagar, J. Kothari, D. Kadikar, Raturaj and Chakravarthy, "SDN Enable Packet Based Load Balancing Technique in Datacenter Networks", *ARNP Journal of engineering and Applied Sciences*, Vol. 12, pp. 4762-4768, 2017.

52. Askar and Shavan, "Adaptive Load Balancing Scheme for Datacenter Networks using Software Defined Network", Journal of University of Zakho, Vol. 4, 2017.
53. U. Zakia and H. B. Yedder, "Dynamic Load Balancing in SDN Based Data Center Networks", 8<sup>th</sup> IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, pp. 242-247, 2017.
54. S. Kaur, K. Kumar, J. Singh and N. S. Ghumman, "Round Robin based Load Balancing in Software Defined Networking", 2<sup>nd</sup> International Conference on Computing for Sustainable Global Development (INDIA COM), New Delhi, pp. 2136-2139, 2015.
55. Morad, Ameer and E. Ali, "Load Balance in Datacenter SDN Networks", International Journal of Electrical and Computer Engineering, vol. 8, pp. 3086-3092, 2018.
56. GitHub, [online]. Available: <https://github.com/mininet/mininet>
57. GitHub, [online]. Available: <https://github.com/noxrepo/pox>.
58. GitHub, [online]. Available: <https://github.com/esnet/iperf>.
59. Linode, [online]. Available: <https://www.linode.com/docs/tools-reference/tools/load-testing-with-siege/>.