Improving the Performance of Software-Defined Network Load Balancer Using Open Flow Based Multi-Controller Topology

A Thesis Presented

By

Kidist Mitiku

To

The Faculty of Informatics

Of

St. Mary's University

In Partial Fulfilment of the Requirements
For the Degree of Master of Science

In

Computer Science

June 22, 2022

ACCEPTANCE

Improving the Performance of Software-Defined Network Load Balancer Using Open Flow Based Multi-Controller Topology

By

Kidist Mitiku Tadese

Accepted by the Faculty of Informatics, St. Mary's University, in partial fulfilment of the requirement for the degree of Master of Science in Computer Science

Thesis Examination Committee:

____

Internal Examiner

____

External Examiner

Melkamu Hunegnaw, PhD [illegible]

____

Dean, Faculty of Informatics

June 22, 2022

DECLARATION

I, the undersigned, declare that this thesis work is my original work, has not been presented for a degree in this or any other universities, and all sources of materials used for the thesis work have been duly acknowledged.

____

Kidist Mitiku Tadese

____

Signature

Addis Ababa

Ethiopia

This thesis has been submitted for examination with my approval as advisor.

____

Dr. Asrat Mulatu

[illegible]

____

Signature

Addis Ababa

Ethiopia

June 22, 2022

ACKNOWLEDGEMENT

First, I gratefully acknowledge to the Almighty of God for the good health and wellbeing that were necessary to complete this thesis. Next, I wish to express my sincere thanks to my advisor Dr. Asrat Mulatu for his invaluable sincere advice and guidance to complete my thesis. Besides, I appreciate my Family, especially my father, mother, and my brother, for encouraging me to be successful in finishing the research and always be by my side for any help that I require

Finally, my due thanks go to my boyfriend Bereket Fitsum, who has constantly encouraged, and provided me helpful ideas.

I also place on record, my sense of gratitude to one and all, who directly or indirectly have put their hands in this venture.

TABLE OF CONTENTS

LIST OF ACRONYMS

API	Application Program Interface
SDN	Software Defined network
CPU	Central Processing Unit
CRM	Customer Relation Management
RAM	Random Access Memory
HTTP	Hyper Text Transfer Protocol
IT	Information Technology
IP	Internet Protocol
ID	Identification
DNS	Domain Name Systems
QoS	Quality of Service
IEEE	International Electronic and Electrical Engineers
GUI	Graphical User Interface
OVS	Open Virtual Switch
DCN	Data Centre Networks
SATCOM	Satellite Communication Networks
SAT	Satellite
MB/sec	Megabyte per second
OF	Open Flow
SLA	Service Level Agreement
GUI	Graphical User Interface
IANA	Internet Assigned Numbers Authority
ToS	Type of Service
TCP	Transmission control protocol
UDP	User datagram protocol
ODL	Open day light

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Due to the emergence of internet of things and 5G networks there's an enormous pressure on the underlying communication networks in terms of demand, performance requirements and dynamic management. To manage incoming traffic, a load balancing technology is assigned to server clusters. In traditional networks routing protocols forward traffic in keeping with the shortest path to reduce cost. This might result in abnormal distribution of traffic causing overloading of communication links. Recently Software Defined Networks (SDNs) has become increasingly popular and potential candidate to beat traditional networks limitations. Software Defined Networks (SDNs), described by an ideal separation of the control and data planes, is being approved as a distinct paradigm for complex network management. In this research work, performance analysis is performed on random, round robin, weighted round robin and least load balancing algorithms in terms of response time /sec transaction rate (trans/sec) and throughput (MB/sec). Moreover a new Open Flow Model based Multi-Controller Topology is proposed and, the proposed topology is able to reduce the response time (sec) by an average of 30.12%, increase the transaction rate (trans/sec) by an average of 39.44% and also increase the throughput (KB/sec) by an average of 10.56% when compared with a single controller topology using random load balancing algorithms in SDN POX controller.

***Keywords*:** Software Defined Networks, Load-balancing Algorithm, Open Flow Model, Multi controller.

CHAPTER ONE

1. INTRODUCTION

1.1. Background

Traditional networks suffer from network management issues because network hardware devices, like switches, routers, and Load balancers, are all vendor specific. For that reason, those devices have tightly coupled control and data planes. So, it is not easy to change their functionality without using a vendor specific network management system [1].

To mitigate the shortcomings of traditional networks, researchers have proposed solutions like software defined networking (SDN) [2], [3], [4] and [5]. As an innovative networking technology that provides (logical) centralization of programmable network control, SDN [6] has been tried to several load balancing and traffic engineering systems [7] SDN allows the flexibleness for one to style and implement own load balancing strategies.

Algorithms vary widely, looking on whether a load is distributed on the network or application layer. Effectiveness of Load distribution mechanism, performance and business continuity can be affected by Algorithm selection. Optimizing network performance in SDN, like several other network, is the need to seek out the foremost load balancing strategies [8].

1.2. Statement of the problem

Load balancing technology is used to ensure better traffic management in server clusters by assigning servers using various server cluster load balancing algorithms. Load balancing performance has become a necessity in networks because of continuous traffic volume increase, user demand growth, and applications' complexity. So, it is important to enhance the performance of the Load balancer. Researcher in [13] evaluates, the performance of load balancer algorithms like round robin, weighted round robin and least load server cluster load balancing algorithms using software defined networking open Flow model by providing Ethio telecom's data enterprise shop customers relation Management system users' as an input.

But still the performance of SDN based load balancers needs to be enhanced. Major research challenges stated in [14] with currently available load balancers are, Poor system performance due to poor traffic load management in the controller and the problem of unmanaged heterogeneous traffic in the network. To alleviate this problem this study targeted to evaluate the performance of load balancing algorithms like random, round robin ,weighted round robin and least load balancing Algorithms in order to identify a better load balancing algorithms in terms of three (3) network performance parameters such as response time/ sec, transaction rate (trans/sec) and throughput (MB/sec) in SDN network. And also deal with improving the performance of SDN based load balancers.

1.2.1. Research Questions

This study attempts to address the following basic research questions.

- Which load balancing algorithm is better in terms of network performance parameters such as response time (sec), transaction rate (trans/sec) and throughput (MB/sec) in SDN Network?
- Which SDN model is used for creating a multi-controller topology?
- Which SDN network topology is better for having a better load balancing performance?
- What results are obtained when the current research result is compared against the prior related research works done on SDN load balancing performance?

1.3. Objective of the Study

1.3.1. General Objective

The general objective of this research is improving the performance of SDN load balancer using Open Flow based multi-controller topology.

1.3.2. Specific Objectives

Based on stated general objectives the following specific objectives are presented.

- Analyse the performance of SDN load balancing algorithms based on existing network topology.
- Review existing related works and techniques.

- Propose a new topology that improves the performance of SDN load balancing algorithms in terms of response time (sec), transaction rate (trans/sec) and throughput (KB/sec).
- Compare the result obtained using proposed multi-controller topology against the Single controller network topology.

1.4. Scope and Limitation of the Study

1.4.1. Scope

The Scope of this study will be SDN architecture with open flow model and analysis of four load balancing algorithms namely Random, round robin, weighted round robin and least load balancing algorithm which is going to be simulated on mininet 2.3.0 and run on POX controller and connected on OpenVswitch with Open Flow Protocol 10 and the load is tested by using siege 3.0.8 tool based on generating ten to seventy (10-70) concurrent requests of clients to 6 servers and finally the performance parameters that are used for the evaluation are response time (sec), transaction rate (trans/sec), throughput (MB/sec).

1.4.2. Limitation

This Work is only limited to POX Controllers in which only the Open Flow model is used to connect a multiple POX controller on a single switch. And only ten to seventy (10-70) concurrent client requests are used to test the load on the Siege Tool, in order to answer a research question.

1.5. Significance

Due to the fast growth in usage of network-based services such as customer relation management system, domain name system, email, etc., there is a huge demand on server clusters. For effective network traffic management load balancing algorithms are used in order to distribute the incoming traffic load to a number of servers to get a better response from servers. Analysis of load balancing performance is necessary to choose appropriate algorithms in terms of a better response time, transaction rate and throughput to increase capacity (concurrent users) and reliability of applications. A better load balancing algorithm improves the overall performance of applications by decreasing the burden on servers associated with managing and maintaining application and network sessions, as well as by performing application-specific tasks.

1.6. Methodology

In order to achieve the objective of this research, two types of research methodology is used,

SDN open flow model as a system model, simulation and analysis of results.

System model:

- The system Model used in this study is SDN Open Flow Model
- A southbound interface between the control layer and Infrastructure Layer of SDN Open Flow Model Used is Open Flow Protocol.
- Python programming language Application Programming Interface (API).

Simulation: The simulation environment that has been used in this research is,

- Desktop with Intel(R) Core (TM) i3-3220 CPU@3.30GHZ,10.0GB RAM which has 64-bit operating system
- Oracle virtual box 6.0.24 is used as a hypervisor to run Ubuntu 16.04 Linux operating system.
- SDN POX controller have installed inside Ubuntu 16.04 Linux operating system
- MININET 2.3.0 is used to emulate the network
- SIEGE load test tool Version 3.0.8 is also used.
- XMING allows the use of Linux graphical applications remotely.
- Ten to seventy (10-70) concurrent client requests are generated by SIEGE load test tool.

Analysis of results: The results and observations from the simulation are analysed and the performance of the SDN load balancing algorithms are evaluated and reported. The performance parameters that are used for the evaluation are response time (sec), transaction rate (trans/sec), throughput (MB/sec). Then a better load balancing algorithm is selected for further analysis and finally using multiple pox controllers on a single switch with different port which is a new topology, re analysis of load balancer performance is done and the result and comparison is reported in the form table and graphical representation.

1.6.1. SDN Open Flow model

Among the four models of SDN (SDN Open Flow model, SDN over relay model, hybrid model and API SDN), SDN Open Flow model is chosen to be investigated in this study due to it provides network programming ability from the centralized view through the modern and extensible API and, it separates the control and data planes.

1.6.2. Mininet Emulator

Mininet is an emulator that allows researchers to deploy large networks on a limited resources like one computer or virtual machine for the use of SDN network and Open Flow. And also mininet provides a realistic service with a minimum cost.
In addition, it enables running unadulterated code collaboratively on virtual hardware using a simple personal computer.
The other possible way to simulate an SDN networks is using hardware test bed which is accurate and fast but very expensive and shared. And also the other option is using simulator which is cheap but every so often got slow and requires code alteration. Mininet provides ease of use, performance accuracy and scalability [16].

1.6.3. Software defined controller: POX

POX is an Open Flow controller which is developed using python programming language that is essentially built to provide a coherent and easy environment in SDN network for investigation and survey. POX controller depends on a component-based model in which some of ongoing work to assist building of the emerging SDN platform in which the whole network components and activities are acknowledged as discrete components that are able to be isolated and utilized when it is necessary.
POX can be applied in different fields such as distribution prototyping and exploration, SDN debugging, network utilization, controller design and programming models [17].
POX controllers are used as an abstraction layer between network application and the network infrastructure.

1.6.4. Load testing tool: Siege

Knowing how much traffic a load balancer server can handle when under stress is essential for planning the future improvement of load balancers. By using tool called siege, it is possible to run a load test on a server and see how the system performs under different circumstances.

Siege is used to evaluate the amount of data transferred, response time, transaction rate, throughput, concurrency and how many times the server returned responses. Siege offers three modes of operation: Regression, internet simulation, and brute force [18].

Towards this research evaluation, internet simulation of siege tool is used for concurrent users

1.6.5. Miniedit

Miniedit is a simple GUI editor for Mininet. And it allows you to simply drag and drop hosts and switches. And also, Miniedit is an investigational tool designed to indicate how Mininet can be enlarged.

1.6.6. Xming

Xming is an open-source application which is used to access a Linux graphical application remotely.

1.7. Related Works

According to [24], SDN is very flexible, enabling the installation of company-defined software based on White box switch. It further allows the configurations of infrastructures to fulfil the network requirements and decrease expenses related to deployment and management.

According to researcher in [26], SDN load balancer overcomes various limitations such as cost and flexibility of traditional load balancers which are caused by the usage of dedicated hardware's. Unlike the traditional load balancers in SDN the Open Flow device is converted into a powerful load balancer by programming its controller. As in many cases with the help of most commercial load balancers, load balancer can also be a single point of failure and to eliminate those problem in future the study suggests using multiple controllers instead of single controllers. One of the showcases for this recommendation the study put an exemplary scenario of controller failing. In this case using multiple controllers helps the machine to take over the role of failed controller and continue routing traffic.

A study [13] simulated Round robin, weighted round robin and least load server cluster load balancing algorithms using SDN Open Flow model in open source POX controller and Open Flow switches in mininet for topology creation on the principles of SDN. The obtained results have compared in terms of network performance parameters: response time (sec), transaction rate (trans/sec) and throughput (MB/sec) by increasing the number of ethio telecom enterprise shop customer relation management system users' as input data. Hence, the researcher concluded that Round robin algorithm is better than weighted round robin and least load server load balancing algorithms in terms of response time (sec), transaction rate (trans/sec), throughput (MB/sec)**.**

In addition, in [13] the research uses a simulation topology with three(3) clients and three(3) servers using mininet in the infrastructure layer with the open flow switch, software defined POX controller connected to the open flow switch using open flow protocol in the south bound interface of the SDN open flow model. Moreover, clients send ten to seventy (10-70) concurrent requests through the internet and the servers are connected to the open flow switch.

Another study [17] investigates the impact of increasing the workload requests from 0 up to 180 requests per second (req/sec) in order to explore average network throughput under the implementation of static, dynamic load balancing algorithms by the POX controller. The study mainly focused on the utilization of HTTPerf by considering that it provides a flexible facility for the generation of various HTTP workloads as well for the measurement of server performance. And the research put a finding that as the number of requests increases the throughput increases as well. Dynamic least bandwidth-based load balance scheme has shown a remarkable improvement in terms of average network throughput up to 8%, 3.3% and 2.56 %, as compared with static load balancing schemes like random, round-robin and weighted round-robin. However, Dynamic least bandwidth recorded a slight ineffective progress. Less than 1 % was recorded when a comparison was conducted with dynamic least connections, this directed the researcher to the fact that their performance was almost the same.

Moreover, a research work in [25] puts Software-Defined Networking (SDN) as a key factor to improve traffic distribution and Quality of Service (QoS) in large scale networks. However, load balancing is the main technology area that must be efficiently implemented associated with efficient resource management and utilization challenges. In this work, four topologies have been proposed: 2Q1L, Multiple, 1Q2L, and 2Q2L. The proposed topologies are simulated using the OFSwitch13 module of the ns-3 network simulator in different scenarios. All proposed topologies outperform the Chavez topology [28] in terms of load balancing However, 2Q1L topology presents the best performance in different simulation scenarios. Besides, the simulation results determine the threshold values of effective parameters leading to the best load balancing on the servers. These variables are the number of flow tables and switch CPU processing capacity. Therefore, the proper selection of the effective parameters can provide satisfying performance with minimum cost.

Another study [31] undergoes a performance comparison of two python-based Software Defined Network (SDN) controllers i.e. POX and Ryu under different network topologies such as Single, Linear, Tree, Dumbbell, Data Centre Networks (DCN) and Software-Defined naval networks which use satellite communication systems (SATCOM) i.e. SDN-SAT [32]. Laboratory results, validated through Mininet has clearly indicated that Ryu has remarkable performance i.e. A TCP throughput increase of 25.56%, 282.54%, 44.85%, 19.88%, 45.45% and latency decrease of 93.48%, 99.96%, 99.90%, 97.08%, 99.33% in single, linear, tree, dumbbell and DCN topologies respectively. Similarly, in SDN-SAT topology Ryu has 0.21% increase in TCP throughput and 34.62% decrease in latency as compared to POX controller.

A novel algorithm also proposed for the purpose of load balancing for an SDN-based datacentre [33]. In this work, mininet emulator has been implemented for the purpose of emulating the proposed system, the suggested algorithm added to the POX controller. To evaluate the algorithm, the study simulated a datacentre with a Fat-Tree topology (k=4). The algorithm proposed to dynamically balance the load by means of re-routing utilizing the information at the SDN controller. The network performance evaluated in terms of throughput, loss, and received data size with and without applying the proposed algorithm. Accordingly, results showed that the proposed algorithm outperforms the traditional load balancing scheme as follows; improves the throughput by a minimum of 21.9%, reduces the loss by 88.2%, and increases the received data size by 20.8 %.

The researchers in [34] proposes, the application of SDN-based Load Balancing by implementing predefined servers in the server-farm that receive the arrived Internet Protocol (IP) data packet from multiple clients with equal amount of loads and executes orders for each server. As a showcase, experiments have been conducted using Mininet and based on several scenarios as follows:

Scenario A, the topology is based on a simple design, which consists of four clients and two servers. The controller was created using an Open Flow controller. The topology consists of a switch controller, two servers and four hosts and gives a high throughput,

Scenario B (The second scenario consists of four clients, and there are four server pools connected to the switch controller, which is the Open Flow controller. Each host is located at a dedicated server, which helps to increase the output performance).

And Scenario C (The third scenario consists of one switch controller connected to the four server pools, which support eight clients. The design is like the previous two topologies, but there is an additional number of clients. The load balancers lag in these design topologies where the response time and latency increase the load balancers) of network topologies in which each uses several servers and multiple clients connected to the servers to produce a different outcome, which is evaluated based on the delay, jitter and throughput parameters.

Finally, findings indicated that scenario A scenario B and C Produce a low jitter value and scenario C produces the lowest delay. SDN results a multi-path direction to reach the best route for a remarkable network performance.

Another researcher in [35] has proposed, a fabric topology to provide a flexible data-centre network. With SDN controllers and has the potential to address critical issues such as bandwidth utilization and network capacity in datacentre networks. In addition, the proposed network fabric consists of various key components: SDN controller, commoditized switches, and host machines. All switches arranged to form a switch pool and connected to a certain number of switches based on performance requirements.

The SDN controller acts as a centralized manager by connecting the switch pool and collects network statistics at run-time, in addition, the host machines connected to switches via their Ethernet interfaces, and this makes fabric topology a critical part of proposed networks. According to this the major physical devices in such networks are low-cost and commoditized switches in relative and there is no definite hierarchy in the proposed network. The flat architecture gives datacentre networks more flexibility in bandwidth utilization and packet re-routing. Moreover, the network fabric enables the switches to have acceptable amount of workload.

Researcher in [36] has evaluated the performance of the SDN load balancers and compared the latency as well as the response time for the design topologies. There is slight time difference on different topologies.

The goal of this project is to design an efficient load balancer using SDN; the design topology consists of the SDN-switch and an Open Day Light (ODL) controller. The flow table holds packet entries which are recorded in the data plane. This project demonstrated a regulated controller using ODL by separating the data plane and the controller. In addition to this the requests from different clients will be directed to various predefined servers in the Round-Robin fashion. This project implements load balancing using the SDN controller and results reduced response time as well as latency successfully.

According to the study the evaluation attributes are latency and response time. These two important factors determine how the SDN load balancer's function when there is a complexity in design topology. In this project, different design topologies are analysed to demonstrate the SDN load balancer functionality. To evaluate the performance the design topologies are compared with and without using Open Flow. In each case, the number of requests is increased for different design topologies and output time is noted. Similar designs topologies are considered without using Open Flow and output times for requests are noted.

1.7.1. Summary of related works

Table 1-1 Summary of related works

Authors	**Purpose of the study**	**Methodology**	**Performance metrics**	**Topology used**	**Result and observation**
Wubishet	Performance analysis of round robin, weighted round robin and least load balancing algorithms	SDN Open Flow Model	Response time, transaction rate, throughput	3 clients, 3 servers, one Open Flow switch and one POX controller	Round robin algorithm has a better performance
Haeeder Munther, Mahdi Nsaif	Evaluating the operational performance of POX controller for SDN environment	SDN Open Flow Model	service Delay, utilized bandwidth, latency and throughput	One POX controller connected to eight switches and eight clients	Recommendation of using POX controller for a better specified performance metrics.

				connected to them.	
Hamid Nejadnik, Rasool Sadeghi, Sayed Mahdi	Study of various topologies with the control placement problem influencing load balancing solutions	SDN Open Flow Switch 13	number of flow tables and Switch CPU processing capacity.	four topologies have been proposed: 2Q1L, Multiple, 1Q2L, and 2Q2L	2Q1L topology presents the best performance in different simulation scenarios in which the topology consists of three controller, three switches , Two servers and three clients,
Mohammad reza Ashouri and Shirin Setayesh	Enhancing the Performance and Stability of SDN Architecture with a Fat-Tree Based Algorithm	SDN Open FLow methodology	throughput, loss, and received data size	a Fat Tree topology with 64 hosts, 8 switches, and a central FloodLight controller	central distribution controllers are used to balancing the load Between network components.
Omran M. Alssaheli, Zainal Abidin, Zakaria, Abal Abas	SDN based Load Balancing for Network Performance Evaluation	SDN Open Flow Model	throughput, delay and jitter	Three different topologies are used for analysis	The topology, which consists of four clients and two servers. With one controller shows large number pf throughput as the number of clients increase.
Venkatesh Kodela	Improving Load balancing mechanisms of SDN using Open Flow	SDN Open Flow Model	Latency and response time	Three designs Are considered to test the efficiency of the load balancers with and without open flow model.	proves that the use of Open Flow Performs load balancing more efficiently.

1.8. Thesis Organization

This thesis is divided into 6 chapters. The thesis is introduced in the first chapter. It includes Background information, a statement of the problem, objectives, methodology, literature review and related works which studies more about SDN Open Flow Model and related works proposing topology improve the SDN load balancing and summary of related works are also included in chapter one. Chapter two deals with The SDN Open Flow Model and detailed Architecture of SDN Open Flow model. The third Chapter is all about analysis of existing SDN load balancers focusing on analysis of load balancing algorithms and analysis based on different topologies. Chapter four deals with the proposed Open Flow based multi-controller topology, the proposed load balancing algorithm and the network performance metrics that are going to be used for evaluating the performance of load balancing algorithms.

Chapter five shows the performance analysis of four load balancing algorithms and three different topologies in which one of the topologies is the proposed multi-controller topology. The simulation setup and the comparative analysis of results are also discussed in this chapter.

The last chapter is chapter six, and it deals with conclusions, contributions, and recommendations for future works.

CHAPTER TWO

2. SDN OPEN FLOW MODEL

2.1. OVERVIEW OF SOFTWARE DEFINED NETWORKING

Software-defined networking (SDN) architecture is designed to make network management easier and more flexible. SDN centralization of management is achieved by abstracting the control plane from the data forwarding function in the discrete networking devices.

An SDN architecture delivers a centralized network based the following components.

- **A controller**, the basic component of an SDN architecture, that enables centralized control, automation, and control, automation, management, and policy enforcement across physical and virtual network environments
- **Southbound API**s transmit information between the controller and the individual network devices (such as switches, access points, routers, and firewalls
- **Northbound API**s transmit information between the controller and the applications and policy engines, to which an SDN looks like a single logical network device [9].

SDN architecture is composed of **three layers**, the **infrastructure layer**, the **control layer**, and the **application layer**, which is illustrated in Figure1.1 Each of these layers performs specific functions and interacts with each other using interfaces.

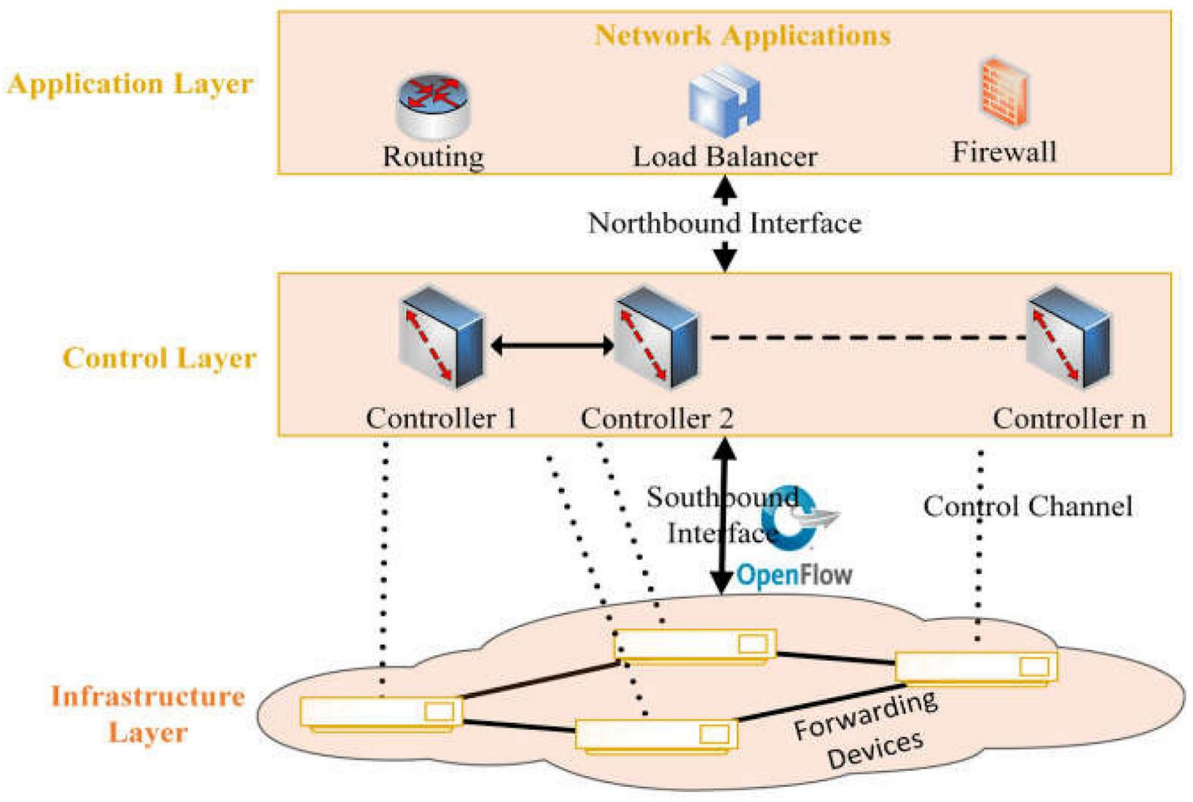


Fig 2-1 SDN Architecture [10]

a) Infrastructure layer

Starting with the infrastructure layer, it has all the physical network elements like a switch, router, OpenVswitch wireless access point etc. These devices' primary functions are to receive the request from the client and forward the data to the next layer, i.e., to the control layer. The data plane of these network components moves it to the controller in the flow tables by following the rules. The controller is the key element to define and install the rules in the switches [11].

b) Control Layer

The control layer is an intermediary between the lower and upper layers of the architecture. SDN controller is a decision-making module in this layer to balance the load to improve performance. It is also responsible for configuring, managing, and controlling the network elements by communicating with them using flow messages. It offers an abstract and centralized view of the layer of infrastructure [11].

c) Application Layer

In this layer there are all the end-user applications with their network requirements, and they use a north bound interface to communicate with the control layer. There are different north bound interfaces and some of them are pyretic, REST API, Frenetic, Procera etc. SDN also allows full network access via one programmable controller, irrespective of whether the network is within the cloud or physically present. The interaction between the networking device and also the controller is completed through Open Flow messages. Flow messages are of various categories.

- **Status:** Controller checks the status of network devices like flow _status, port_ status, queue _status, group_ status and table_ status.
- **Connection**: An echo request and reply message are exchanged between the networking device and controller to verify whether the controller is active or not.
- **Asynchronous**: Asynchronous messages reach the controller via the networking system from the configured switch to remove the flow rule from the networking device, configure and apply failure, port up / down [11].

2.1.1. Expected benefits of SDN

SDN Network is expected to provide visibility over the network state and enable service assurance. And also minimize administration overhead of managing networks.in addition it supports agility to adapt and adjust according to the need of administrators using a centralized controller.

Other expectations of SDN can be

- Rearrange the network according to the need or dynamically to provide services or meet defined Service Level agreement.
- Configure the network to allow or deny traffic patterns (i.e., traffic steering).
- Configure the network to meet the demand of new workloads, and automatically allow cross-workload communication.
- Remove the service particular network configuration when it is disarmed, and re arrange affected network elements accordingly [15].

2.2. SOFTWARE-DEFINED NETWORK

Software defined networking (SDN) architecture allow a centralization of network control and management. And also provides an easy scaling and change implementation. There are four types of SDN networks are defined below:

- **Open SDN:** in open SDN, virtual and physical devices which are responsible for routing the data packets are controlled and managed by open protocols.
- **API SDN:** Use southbound API programing interface In order to control the flow of data to and from each device.
- **Overlay Model SDN:** It is an SDN and network virtualization method, which allows running a logically separate network on existing network.
- **Hybrid Model SDN**: SDN and traditional networking are combination of Hybrid model SDN, allowing the optimal protocol to be allocated for each type of traffic and is often used as a phase-in approach to SDN [29].

2.2.1. Open Flow

Open Flow (OF) is the former software defined networking standard that can better adapt to changing business requirements. And open flow allows the forwarding plane of network infrastructures such as switches and routers, both physical and virtual (hypervisor-based) to directly interact with SDN Controller.

In SDN network, SDN Controller is like a brain, delivering information to network devices such as switches and routers via southbound APIs and the applications and business logic via northbound APIs.

SDN controller casts down changes to the network devices like switch/router, flow-table allowing network administrators to partition traffic, control flows for optimal performance, and start testing new configurations and applications through Open Flow protocol. In which any device that needs to communicate with SDN controllers in an OF environment must support Open Flow protocol [27].

The open flow explain both the communication protocol between SDN control plane and the SDN data plane, and also the behavioural part of data plane. The open flow protocol also explains the message format that is interchanged between device (data plane) and control plane (controller).in addition the open flow defines how the device should react and respond to commands from the controller in various situations [30].

2.2.1. Benefits of Open Flow-Based Software-Defined Networks

The open flow-based SDN technologies allow IT to address the dynamic nature of today's applications, high-bandwidth, adapt the network to ever-changing business needs and significantly reduce management complexity and operations.

The benefits that carriers and enterprises achieve through an open flow based SDN architecture include:

- **Centralized control of multi-vendor environments**: SDN control software can control any open flow enabled network infrastructures from any vendor, including virtual switches, routers and switches. Rather than controlling groups devices from individual vendors, IT can use SDN-based orchestration and management apparatus to quickly deploy, update devices and configure across the entire network.
- **Reduced complexity through automation**: The SDN-based open flow provide good management framework and the resilient network automation, which help in order to develop apparatus that automate tasks that are done manually today. This will help to decrease network instability and reduce operational overhead and support the arising IT and self -service provisioning models.
  Moreover with SDN, Cloud based applications can be managed through provisioning systems and intelligent orchestration in order to reduce operational overhead while increasing business agility
- **Higher rate of innovation:** SDN adoption accelerates business innovation by allowing IT network operators to literally program-and reprogram-the network in real time to meet specific business needs and user requirements as they arise. By virtualizing the network infrastructure and abstracting it from individual network services, for example, SDN and Open Flow give IT- and potentially even users-the ability to tailor the behaviour of the network and introduce new services and network capabilities in a matter of hours.
- **Increased network reliability and security:** An open flow based SDN architecture banish the need to individually configure network devices each time, a policy change or service application is added or removed which minimizes the probability of network failures due to lack of consistent policy configuration.

Because of SDN controllers provide complete visibility and control over the network it is possible for IT to define high-level configuration and policy statements in order that is translated down to the infrastructure through open flow. So that they can ensure that access control, quality of service, security, traffic engineering and other policies are enforced consistently across the wired and wireless network infrastructures, campuses, and data centres.

Consistent configuration minimized operational cost, more dynamic configuration capabilities with minimized error gives a better advantage for Enterprises and carriers.

More granular network control: By decoupling the network control and data planes, open flow based SDN architecture abstracts the underlying infrastructure from the applications that use it. Open flow's flow-based control model allows IT in order to apply policies at a very grainy level, including the user, device and application levels, session in a highly abstracted, automated fashion. So that this control enables cloud operators to support multi-tenancy while maintaining traffic isolation, security, and elastic resource management when customers share the same infrastructure. Allowing the network to become as programmable and manageable at scale as the computer infrastructure that it increasingly resembles.an SDN approach fosters network virtualization, enabling IT staff to manage their servers, applications, networks and storage with a common approach and device set [19].

2.3. OPEN FLOW ARCHITECTURE

Open flow is defined by the open networking foundation in which it supports a multivendor standard and The Open Flow architecture is composed of 3 basic components listed below.

- The Open Flow controller.
- The Open Flow device (Switch)
- The Open Flow Channel.

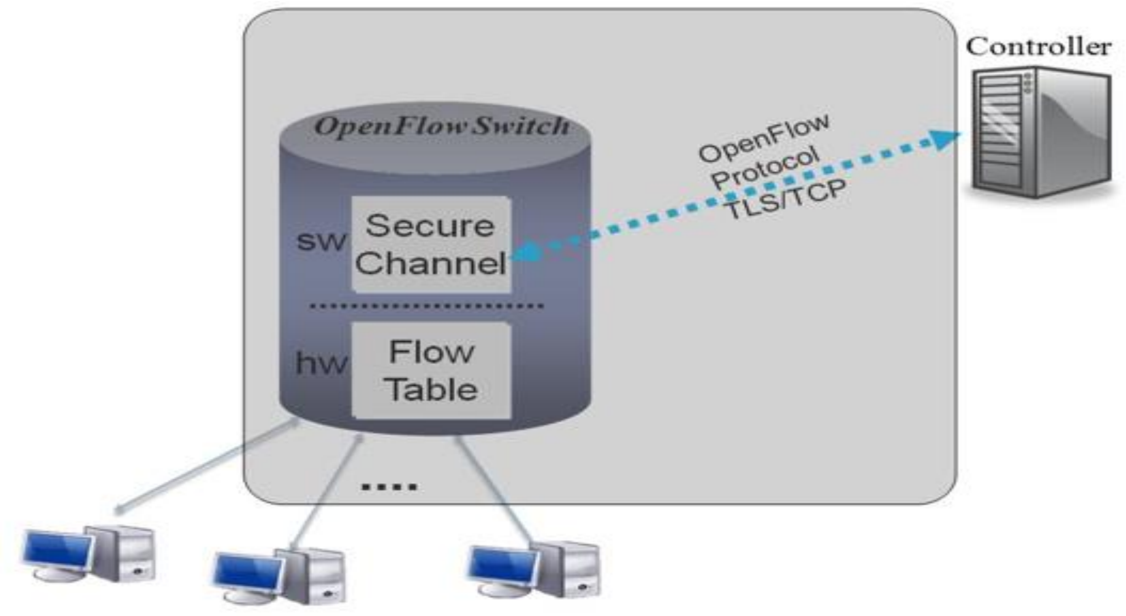


Fig 2-2 Open Flow Architecture [20]

2.3.1. Open Flow Controller

In order to achieve better forward performance in the open flow approach, the network devices should kept simple so that the network control is done by controller. The open flow controller build up all open flow devices, monitors the overall status of the entire network and maintains topology information [20].

The controller can manage flow entries proactively and reactively using open flow protocol. In which e controller controls the switch using the open flow protocol [19].
The Open Flow control plane is different from the traditional control plane in terms of three different cases. The first is the ability to program distinct data plane components with a common Open Flow standard language. The second case is that the control plane and the data plane are on a separate hardware device because of the controller's ability to program data plane elements remotely over the internet, unlike of the legacy switches which exists on the same physical box. Thirdly, the controller's ability to program multiple data plane elements from a single control plane.

programming all the packet-matching and forwarding rules in the switch is the responsibility of The open flow controller Considering that traditional router would run routing algorithms in order to determine how to program its forwarding table. Any changes that result in re-computing routes will be programmed onto the switch by the controller [21]. An Open Flow controller uses the *Open Flow protocol* to talk to Open Flow switches. The interface that connects them is referred to as an *Open Flow channel*.

Certainly, multiple Open Flow channels are possible if an Open Flow switch is managed by multiple controllers. The Open Flow channel can be either encrypted using TLS (Transport Layer Security) or directly over TCP using the default port 6653 approved by IANA [30].

SDN controllers that support Open Flow software includes:

- **NOX:** NOX is a Network Operating System that provides control and visibility into a network of Open Flow switches. It supports concurrent applications written in Python and C++, and it includes a number of sample controller applications.
- **Beacon:** Beacon is an extensible Java-based Open Flow controller. It was built on an OSGI framework, allowing Open Flow applications to be built on the platform to be started, stopped, refreshed, and installed at run-time, without disconnecting switches.
- **Trema:** Originally named Helios, Trema is an extensible Open Flow controller built by NEC in the programming languages of Ruby and C, targeting researchers.
- **NEC Programmable Flow:** Trema is the foundation for the Programmable Flow from NEC. Programmable Flow automates and simplifies network administration for better business agility, and provides a network-wide programmable interface to unify deployment and management of network services with the rest of IT infrastructure.
- **Lumina SDN Controller:** In December 2017, Lumina released the Lumina SDN controller 7.1.0, which supports Open Daylight Nitrogen (the seventh Open Daylight platform). It also has support for Open Daylight's Karaf 4, which allows users to choose the controller's protocols and services.
- **Big Switch:** Its Big Cloud Fabric controller creates a virtual private cloud based on SDN controller abstractions and open networking hardware switches [17].

2.3.2. The Open Flow Switch

The Open Flow Device is any network equipment supporting the Open Flow protocol, such as a switch. Each device maintains a Flow Table that indicates the processing applied to any packet of a certain flow [20].

An Open Flow Switch uses an Open Flow channel for an external controller and perform the packet forwarding and packet lookup using one or more flow tables and group tables [19].

When an Open Flow switch receive a user packet, it is going to be looked over in a flow table in order to check that the packet matches. If it matches, the user packet is examined and forwarded based on the predefined action. Every so often look in on further flow tables as predefined by the actions. If the user packet mismatches the last flow table entry, and no table-miss entry, the packet will be dropped. Attention that a table-miss entry can be thought of as a default entry to let a packet know that if there is no match, then it is to be sent to the controller. Thus, if the user packet match occurs only at the table-miss entry, then the packet is forwarded to the controller for further action. The controller may choose to insert a new flow table entry for this new flow or to drop this user packet. The packet flow in an Open Flow switch is described in Fig. 2-2 [22].

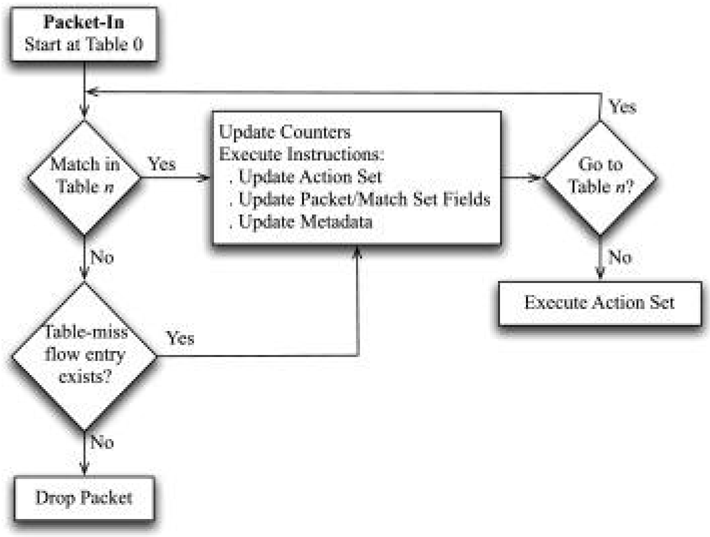


Fig 2-3 packet flow in Open flow switch [22]

2.3.3. Open Flow Ports

Open Flow switches are passing packets between each other by the use of a network interfaces called Open Flow ports. A user packet can be forwarded from one Open Flow switch to the other by only using an output Open Flow port on the first Open Flow switch and an ingress Open Flow port on the second switch.

For Open Flow processing, An Open Flow switch makes a number of Open Flow ports available. The set of network interfaces provided by the switch hardware, may not be the same with the set of open flow ports. Since some network interfaces might be disabled for Open Flow, The Open Flow switch may provide extra Open Flow ports [19].

2.3.4. The Open Flow Protocol

The Open Flow protocol is an interface between the controller and the switch that sets up the flow table. And the flow table is managed (add and remove) flow entries by the controller using the Open Flow protocol.

The Flow Table is updated by the controller by adding and removing flow entries using the Open Flow protocol. In order to command the Switch, to apply some actions (Forward, drop or encapsulate) on a certain flow, The Flow Table contains a lot of Flow Entries associated with actions to command the Switch [20].

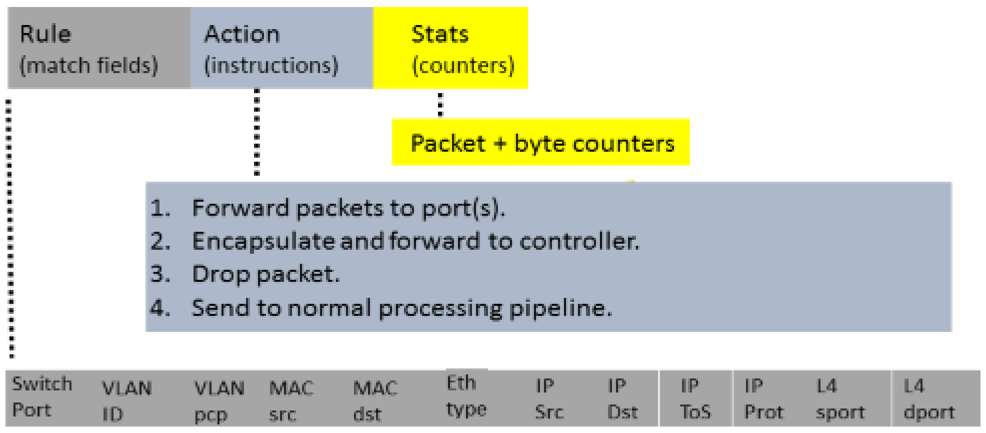


Fig 2-4 Open Flow table [20]

As shown on the above figure 2-3, each Open Flow devices have a Flow table with flow entries In the SDN Open Flow network. The Flow Entry contains three (3) fields which listed below:

- The Header field: it is used to define the match condition to an exact flow.
- The Counter field: are used to count the rule occurrence for management purposes.
- The Action field: define the action to be applied to a specific flow.

When a packet arrives at the Open Flow switch, it will be matched by a Flow Entry in the Flow Table, so the action will be executed if the header field is matched, and the counter is updated. In case the user packet mismatch any flow entry, then the user packet will be sent to the controller. Note that in the Flow Table a higher number means a higher priority. In which the priority is used to match the user packet, whereas the flow entry containing higher priority will be chosen [30].

2.3.5. The Controller-Switch Secure Channel

The communication between the Open Flow controller and the Open Flow devices is secured by the Asymmetrical encryption, which is a Transport Layer security, therefore unencrypted TCP connections are allowed. These connections may be *in-band* or *out-of-band*. The Figure 2-4 below depicts these two variants of the secure channel. In the out-of-band example, we see in the figure that the secure channel connection enters the switch via port Z, which is not switched by the Open Flow data plane. To the secure channel process in the switch, some traditional network stack will forward the Open Flow messages using the secure channel. Where all Open Flow messages are parsed and handled. Thus, the out-of-band secure channel is relevant only in the case of an Open Flow-hybrid switch. In the in-band example, we see the Open Flow messages from the controller arriving via port K, which is part of the Open Flow data plane. In this case these packets will be handled by the Open Flow packet-matching logic shown in the figure. The flow tables will have been constructed so that this Open Flow traffic is forwarded to the LOCAL virtual port, which results in the messages being passed to the secure channel process. Note that it is recommended tot to use the TLS based encryption when the controller and all the switches it controls are located entirely within a tightly controlled environ.ment such as a data centre, because of a performance overhead is incurred by using this type of security [21].

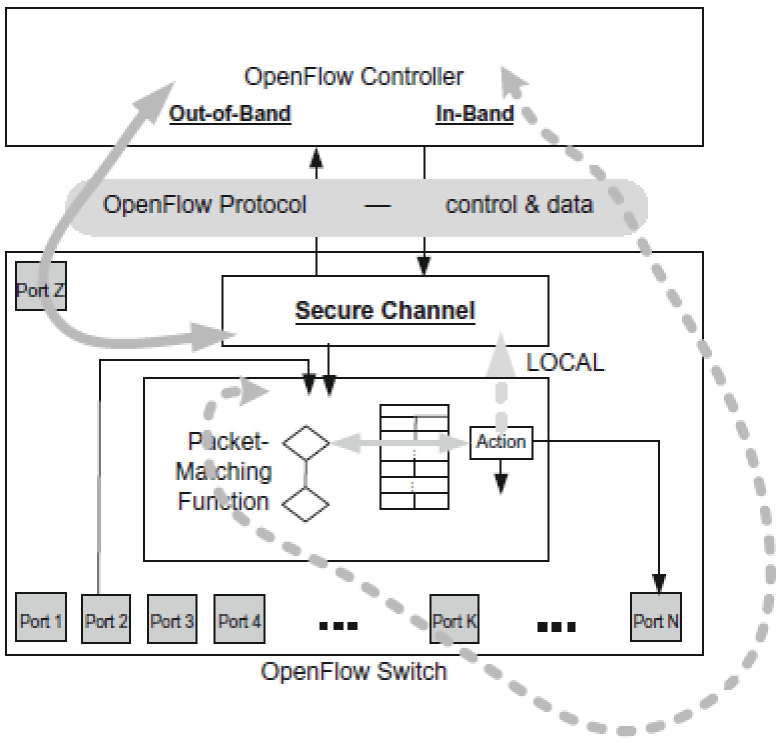


Fig 2-5 Open Flow controller-switch secure channel [21]

2.3.6. Open Flow messages

Open Flow messages are sent and received between the controller and the data paths (Open Flow instances or devices) it manages. These messages are byte streams, the structure of which is documented in the Open Flow Protocol Specification documents published by the Open Networking Foundation (ONF). Open Flow protocol categorize message types in to three, each with multiple categories:

- Controller-to-switch
- Symmetric
- Asynchronous

2.3.6.1. Controller-to-switch messages

Controller-to-switch messages are first launched by the controller and the functionality of this message is controlling and managing the state of the switch. This controller-to-switch messages might not need a response from the switch and are classified in the following categories.

- **Features**

While the transport layer security session is initiated, the controller sends a feature request message to the switch. And then the switch have to reply with features reply message which indicates the features and abilities that the switch can support.

- **Configuration**

Since the controller have an ability to set and query configuration parameters in switch, the switch also can only responds to a query from the controller.

- **Modify-State**

The controller sent the Modify-State message in order to add, delete or modify the flow table entries or to set the switch pot priorities, and also this messages are sent to manage the state of the switch.

- **Read-State**

The read-state messages gather statistics from the follow table of the switches, ports and from each flow entries.

- **Send-Packet**

The send-packet message is used to send packets out of predefined ports on the switch and those messages are sent by the controller.

- **Barrier**

This barrier messages are used to verify whether to receive a successful completion notification or message dependencies have been met and this message is also used by the controller [23].

2.3.6.2. Symmetric messages

Symmetric messages are one of an open flow messages in which the messages are launched by the controller or the switch and they are sent without and request. And there are three subcategories of symmetric messages: -

- **Hello Message**

Hello messages are one of the symmetric messages in which the messages are interchanged between the controller and the switch based on the connection setup.

- **Echo Message**

This types of massaged are used to specify the bandwidth, latency and or the heartbeat of a controller-switch communication. And also this messages, can be sent from the controller or the switch and there have to be an echo reply message.

- **Vendor Message**

This vendor message produces the standard way of an Open Flow switch to give supplemental functionality within the Open Flow message types for future revision of Open Flow [23].

2.3.6.3. Asynchronous messages

Asynchronous messages are one of an Open Flow message in which it is first launched by the switch and used to modify the controller of network events and variations to the switch state.

Switches send the asynchronous messages to the controller to indicate that the packet is arrived, the state of the switch is changed, or packet is error. And four categories of asynchronous messages are defined below: -

- **Packet-in messages**

Packet-in messages are one type of asynchronous message in which these types of messages are sent to the controllers whether packets have a matching entry or not. When the switch have enough space to buffer packets sent to the controller, the packet in message includes A buffer ID and the packet header with a default size of 128 bytes are used by the controller when it is ready to forward the packet for the switch, but if the switches have insufficient buffer space, they have to send the entire packet to the controller as a part of the message.

- **Flow-Removal messages**

The flow entry removal message shows that while a flow entry is added to the switch, by a flow modify message, an idle timeout value shows the time entry should be removed due to inactivity or large timeout value. In which this value shows that when the entry should remove except with activity.

- **Port-status messages**

When the port configuration state changes (the port status changes, port disabled by the user or a change in port status as predefined in 802.1D) the switch is expected to send the port-status message to the controller. Open Flow switches might support spanning tree protocol. When the Port is changed in the case of spanning tree are sent to the controller using the port-update messages.

- **Error messages**

Error message indicate a message that the switch alerts the controllers in the case of problems [23].

2.3.6.4. Packet Matching

While the packet arrives at the Open Flow switch either from an input port or some often from the controllers, the packets are matched in the flow table to specify that there is a matching flow entry. The below listed match fields associated with the incoming packet might be used for matching against flow entries:

- Switch input port
- VLANID
- VLAN priority
- Ethernet source address
- Ethernet destination address
- Ethernet frame type
- IP source address
- IP destination address
- IP protocol
- IP Type of Service (ToS) bits
- TCP/UDP source port
- TCP/UDP destination port

These 12 match fields are collectively referred to as the basic 12-tuple of match fields [21].

2.4. LOAD BALANCING TECHNIQUES IN SDN

Load balancing is a technique that optimize network performance and boost QoS by, assigning load to network elements. Load balancing strategies and algorithms play a big role in improving the efficiency by assigning or transferring the load to support both the service providers and end users. Load balancing helps to forecast the traffic bottleneck prior the occurrence.

- **The Need for Load Balancing in SDN**: as the concurrent requests are increasing from clients, the servers in the network are getting overloaded, therefore, to have a better service and meet the requirements of QoS, the load must be balanced. If this concern is ignored, then it leads to failure of the links and sometimes server crash. In comparison with the traditional networks, the switches have only data planes with them while separating all the control planes from the switches and moving them to a centralized unit called controller in Software Defined Networks [11].
- **Significance of Load Balancing**: The three layers in software-defined networks communicate with each other using interfaces. On the one hand, the network devices present in the infrastructure layer forwards the requests to the control layer. On the other hand, the applications with burdens of various services in the application layer are to be fulfilled. So, to satisfy both the requirements, the control layer plays a central intelligent role. With the increasing demand of the customers over the cloud services, the number of requests from the clients is increasing, so this puts an increased load on the networking elements to handle them. Management of load raises a concern to efficiently balance the load with the existing infrastructure and gain the satisfaction of the customers by improving the QoS provided to them [11].

a) Conventional Load Balancing Techniques

The conventional load balancing techniques are the current techniques in use to balance the load. These techniques use the traditional algorithms for load balancing, and prominent of them include round-robin technique, equal-cost multipath routing protocol, least connections, random techniques, etc. [11].

b). Artificial Intelligent Based Load Balancing Techniques

Artificial Intelligence-based techniques use a metaheuristic approach to solve real-world problems. The sub-areas of artificial intelligence include deep learning, neural network, natural language processing, knowledge representation, reasoning (logical and probabilistic), and decision making with search, planning, and decision theory.

Artificial intelligence-based load balancing techniques provide better learning abilities and foster decision making in SDN [11].

CHAPTER THREE

3. ANALYSIS OF EXISTING SDN BASED LOAD BALANCERS

3.1. Load balancing Algorithms

3.1.1. Round Robin Load balancing algorithm

Round robin algorithm is one of the popular algorithms used in SDN based load balancing techniques. This approach distributes the request to the paths in sequence, starting from the first path to the last one in rotation continuously. It passes new requests to the next server in line and distributes user requests evenly across an array of servers being load balanced. The main advantages of the round. The robin algorithm is simple to implement, cheap, very predictable, fair and works best when all servers have equal capacity. The disadvantage of this algorithm is it has no priority means that it doesn't give any special priority to more important tasks. In the SDN open flow model, when the controller is initialized, the first statistics of all servers' information (IP address, MAC address, ID, port) in the cluster are collected and stored in a Hash Map. When the VIP module is calling the Open Flow enabled Round-Robin algorithm to assign the incoming traffic, the algorithm will determine which server to use according to the last selected server's ID. This method ensures that all servers will be visited in a loop.

3.1.2.Weighted round robin Load balancing algorithm

Weighted round robin algorithm is one of the methods use d in server cluster load balancing. It assigns user requests first by checking the weight of each server in the cluster. The advantage of weighted round robin algorithms is for those datacentres having different servers in speed and memory. In [12], it has been explained that weighted round robin algorithms usually specify weights in proportion to actual capacities. So, here assume that, the Server 1's capacity is 10 times more than Server 2's, then we can assign a weight of 10 to server 1 and weight of 1 to server 2.

3.1.3. Least Load balancing algorithm

The least load server load balancing algorithm is one of the load balancing methods applied on SDN networks. This approach forwards the request to the path that has the least number of current connections.

3.1.4.Random Load balancing algorithm

Random load balancing algorithm is commonly used in SDN networks. This approach randomly allocates the traffic to the convenient servers. In this algorithm, a process can be handled as the node is selected based on a random selection, without having any information about the current, or the previous load over the node. Random load balancing algorithm uses a random number generator, and it is preferred when all processes are equally loaded.

3.2. Performance analysis based on different Algorithms

The researcher in [13] evaluates, the performance of round robin, weighted round robin and least load server cluster load balancing algorithms using software defined networking open Flow model by using Ethiopian telecommunications corporation enterprise shop CRMS users' data as input for performance analysis of different load balancing algorithms in terms of network performance parameters such as response time/sec, transaction rate trans/sec throughput (MB/sec) in SDN network. Simulation has been used as a methodology to evaluate server load balancing algorithms using the Open Flow model by creating a virtual environment with an oracle virtual box. In addition, a mininet simulation tool has been used to create the network topology and POX controller used in the control layer of the open Flow model to do the performance evaluation of the load balancing algorithms. The simulation result shows that the round robin algorithm is better than the weighted round robin and least load server load balancing algorithms in terms of response time (sec), transaction rate (trans/sec), throughput (MB/sec). The following Topology is used for analysis of the performance.

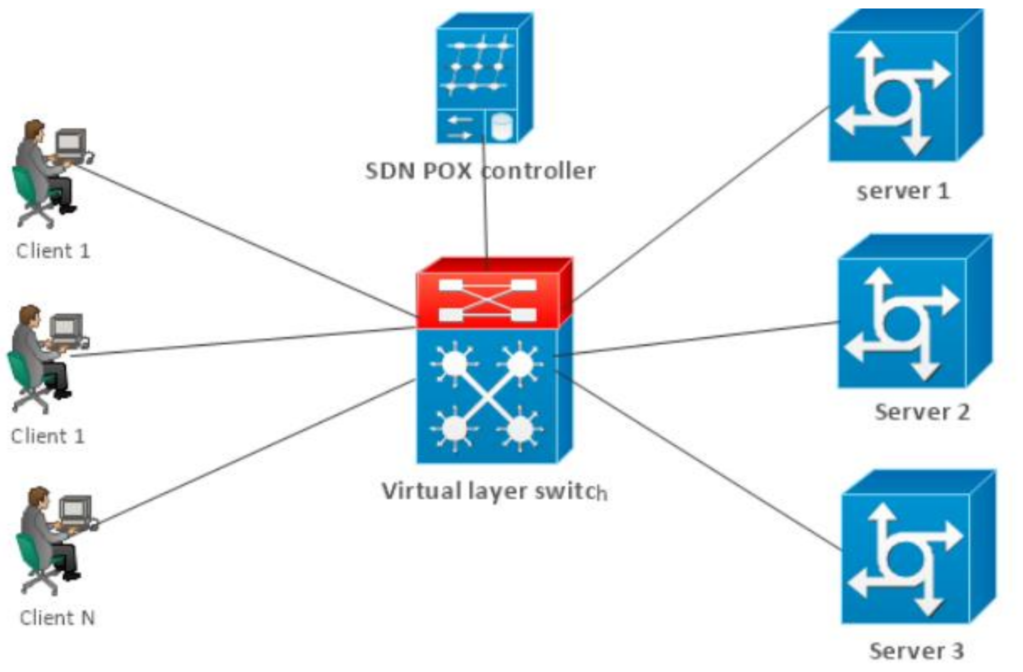


Fig 3-1 SDN based server cluster load balancing topology [13]

- The results have been presented in tabular and graphical form. The response time in (sec) of round robin, weighted round robin and least load server cluster load balancing algorithms by increasing the number of concurrent users starting from 10 concurrent users to 70 concurrent users with gradual increase of 10 concurrent users in between has been presented in figure 3-2 below.

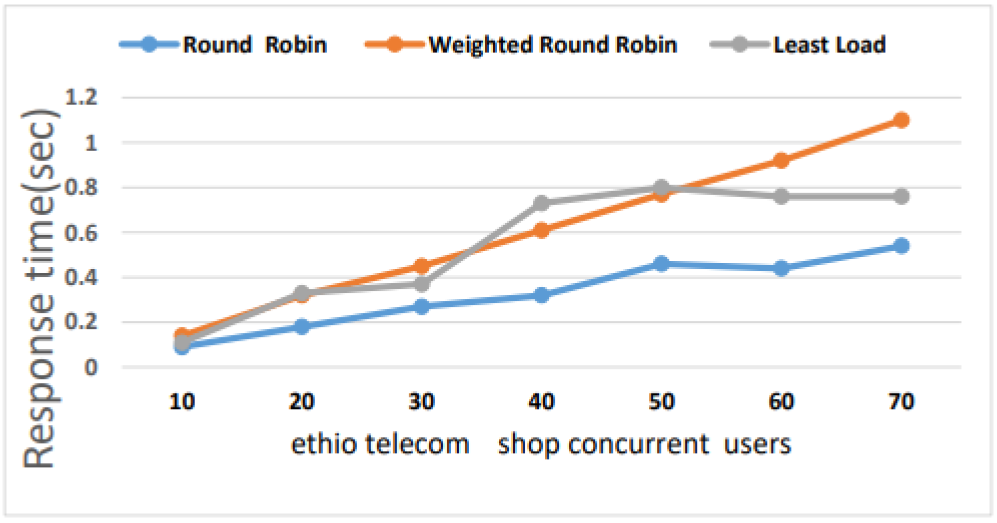


Fig 3-2 Response time graph [13]

As we see in the response time (sec) graph in Figure 3.2, the response time (sec) of round robin, weighted round robin and least load server cluster load balancing algorithms increases as the number of concurrent users increases, but the response time (sec) of round robin server cluster load balancing algorithm is less than when compared to weighted round robin and least load server cluster load balancing algorithm. This is because the round robin server cluster algorithm doesn't check the current performance of the server cluster. It simply assigns requests in a predefined pattern in cyclic fashion.

- The transaction rate in (trans/sec) of round robin, weighted round robin and least load server cluster load balancing algorithms by increasing the number of concurrent users starting from 10 concurrent users to 70 concurrent users with gradual increase of 10 concurrent users in between has been presented on the figure 3-3 below.

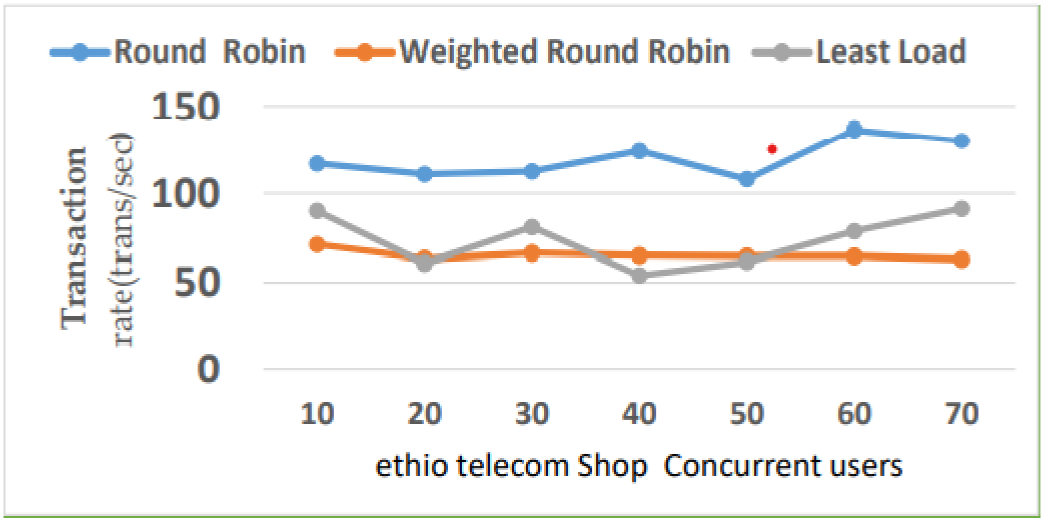


Fig 3-3 Transaction rate graph [13]

When we see the transaction rate (trans/sec) graph in figure 3.3, the round robin server cluster load balancing algorithm has a better transaction rate (trans/sec) as the number of concurrent users increases when compared to weighted round robin and least load server cluster load balancing algorithms. The reason for this is that the response time(sec) of round robin server cluster load balancing algorithm was better than weighted round robin and least load server cluster load balancing algorithms as shown in figure 3.2, and if round robin server cluster load balancing has less response time(sec) ,it can handle more transaction per second.

- The throughput rate in (MB/sec)of round robin, weighted round robin and least load server cluster load balancing algorithms by increasing the number of concurrent users starting from 10 concurrent users to 70 concurrent users with gradual increase of 10 concurrent users in between has been described on the below 3-4 graph.

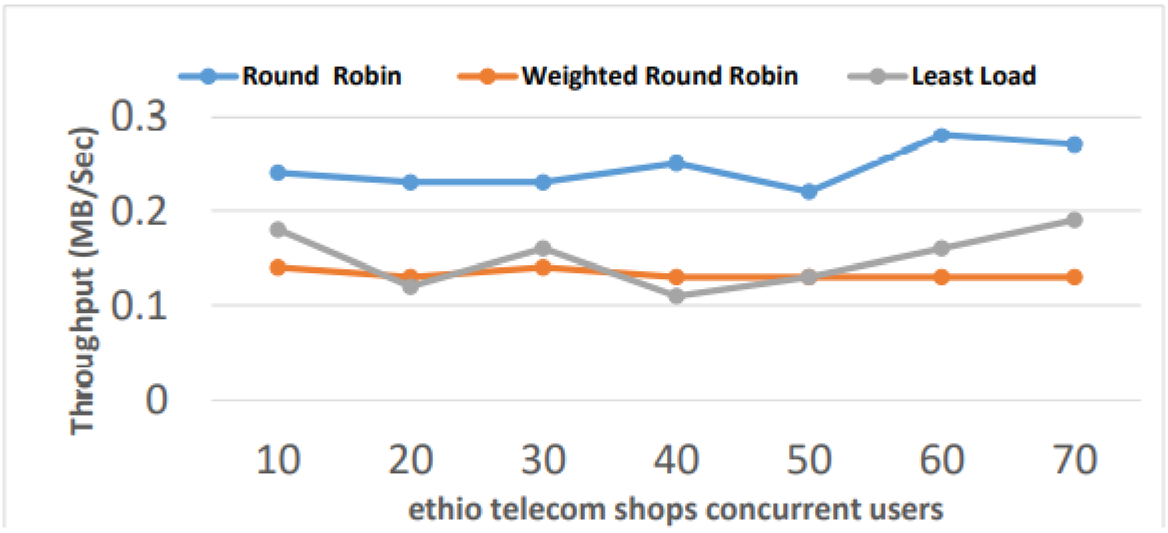


Fig 3-4 Throughput graph [13]

As we see in the throughput (MB/sec) graph in Figure 3.4, it has been observed that the throughput (MB/sec) for round robin server cluster load balancing algorithm is better than that of weighted round robin and least load server cluster load balancing algorithms. The justification for having a better throughput (MB/sec) in round robin server cluster load balancing algorithm is that since it has a better response time (sec) and if it has a better response time (sec), it will have better transaction rate (trans/sec) and also if it has a better transaction rate (trans/sec), it will achieve better throughput (MB/sec).

3.3. Performance analysis based on different Topology

Researcher [36] did a simulation to prove that the SDN with an Open Flow protocol on load balancers performs more efficiently. In this research three design topologies are examined for the analysis of the efficiency of the load balancers.

The design topologies are then compared with design topologies without Open Flow. The evaluation of the topologies is based on response time and latency.

- The first design topology is simple design in which there are equal number of clients and servers, here there are two clients and two servers in the server pool connected to SDN switch and the clients connected to a normal switch. These two clients have to convey with the servers in the server pool. Whereas the servers use the round robin algorithm as a load balancing.

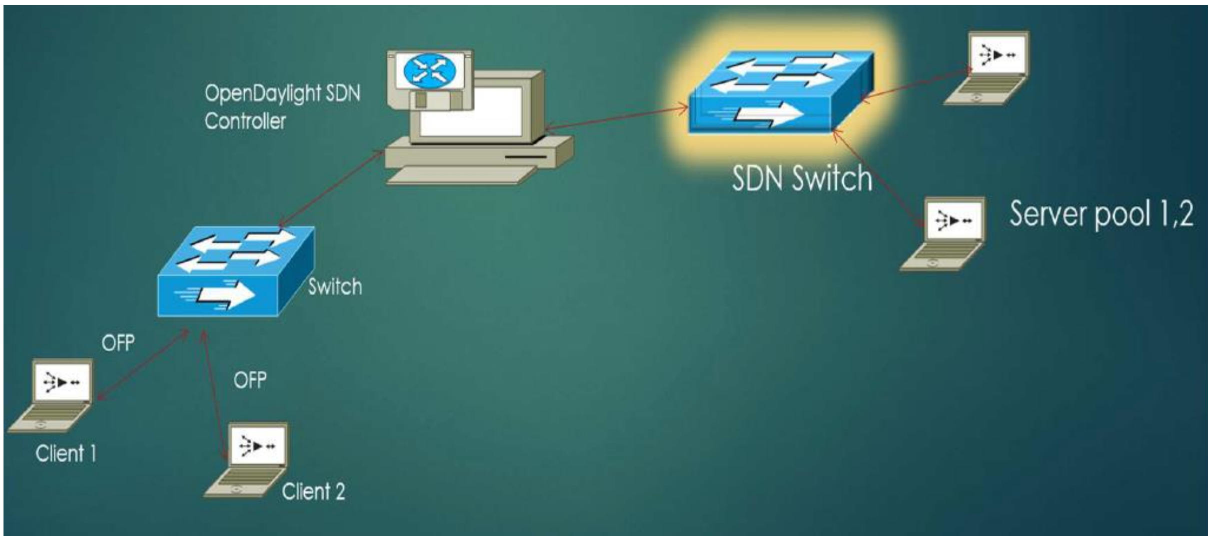


Fig 3-5 Design topology one [36]

- The second design topology is a more complex design where there are five clients, but

There is not an equal number of servers like the first design. In this design, there are five clients but only four servers in the server pool to compensate those five clients. The SDN switch connects the controller and the servers.

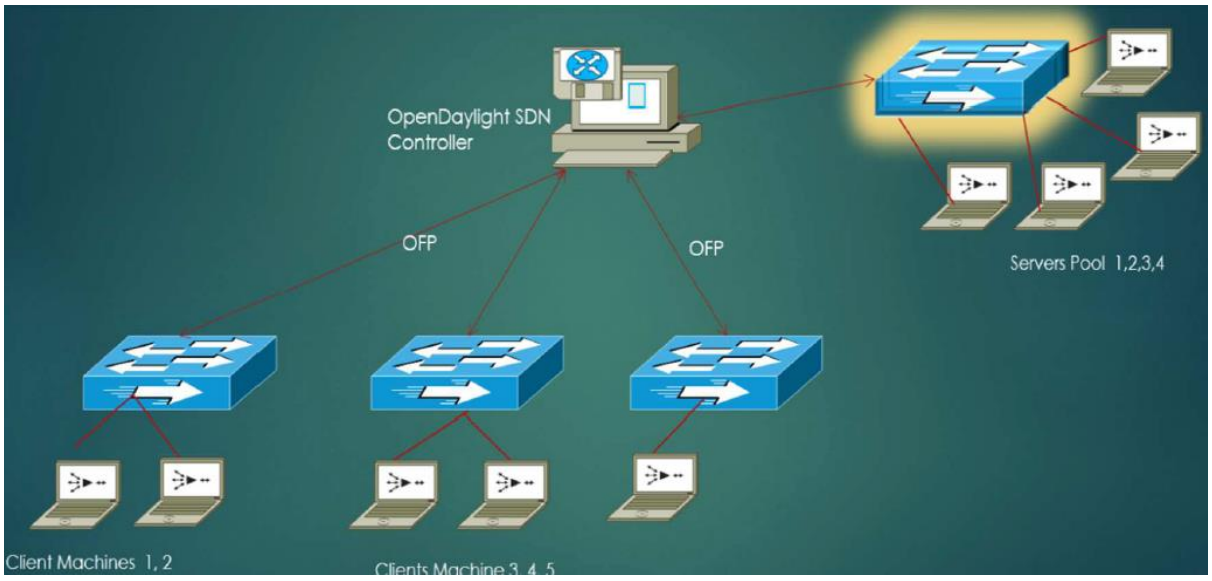


Fig 3-6 Design topology two [36]

The third design topology is a very complex design in which there are two servers in the pool and nine clients.

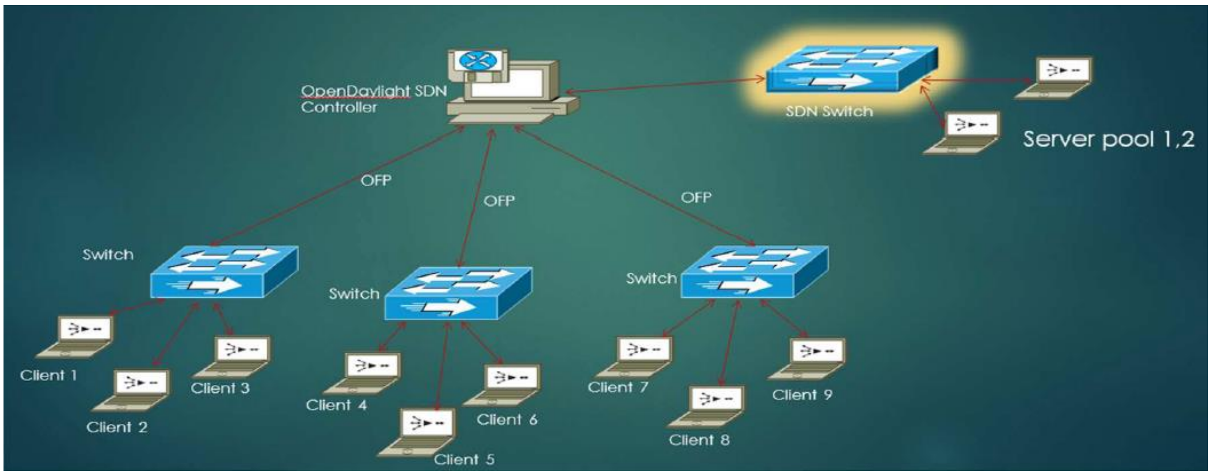


Fig 3-7 Topology three [36]

To evaluate the performance of the design topologies are compared with and without using Open Flow. In Table 3.1, time in seconds is noted when using Open Flow protocol for output times. In each case, the number of requests are increased for different design topologies and output time is noted. In Table 3.2, similar design topologies are considered without using Open Flow and output times for requests are noted.

Table 3-1 Output time for requests using Open Flow [36]

Number of requests	**Topology 1 (Time in seconds)**	**Topology 2 (Time in seconds)**	**Topology (Time in seconds)**
10	0.21	0.24	0.26
20	0.25	0.28	0.28
30	0.28	0.3	0.32
40	0.33	0.34	0.37
50	0.35	0.37	0.398
60	0.36	0.38	0.41
70	0.37	0.384	0.418
80	0.378	0.392	0.424

Table 3-2 Output time for requests without Open Flow [36]

Number of requests	**Topology 1 (Time in seconds)**	**Topology 2 (Time in seconds)**	**Topology (Time in seconds)**
10	0.37	0.397	0.44
20	0.395	0.41	0.49
30	0.42	0.44	0.55
40	0.45	0.47	0.59
50	0.48	0.0.51	0.65
60	0.51	0.558	0.71
70	0.54	0.59	0.78
80	0.59	0.63	0.89

As we can see from the result, the research in [36] proves that the SDN with an Open Flow protocol on load balancers performs more efficiently than topologies without using Open Flow protocol. And also, the researcher in [36] concluded that compared to topology two and topology 3 topology one has a better response time.

CHAPTER FOUR

4. PROPOSED OPEN FLOW BASED MULTI-CONTROLLER TOPOLOGY

4.1. Proposed Network topology

In order to use multiple controllers on a single Open virtual switch, Open Flow model, the most commonly used southbound interface for SDN is used. In this work, The Proposed topology for having a better performance of SDN based load balancer is Open Flow based Multi-controller Topology. which is simulated using mininet in the infrastructure layer with the open flow switch, software defined multiple POX controllers connected to the open flow switch using open flow protocol in the south bound interface of the SDN open flow model on port 6633 and 6630. Moreover, concurrent HTTP requests are sent by the client through the internet and the servers are connected to the open flow switch.

The proposed design produces a better performance compared to topologies proposed by researcher [13] and researcher [36].

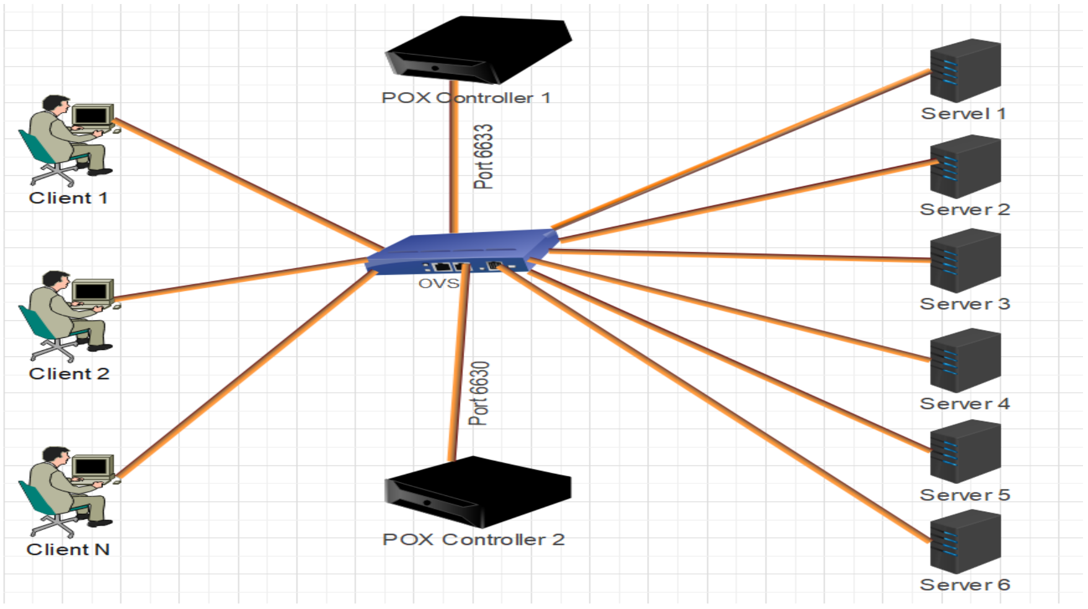


Fig 4-1 SDN based multi-Controller load balancing topology

4.1.1. The workflow for the proposed topology

1. There are three clients and six servers, in which clients send a concurrent HTTP request to HTTP servers.
2. The request is first sent to the Open V switch and the switch is connected to a multiple POX controller.
3. Controllers are connected to the Open V switch through port 6633 and 6630 by the use of Open Flow Model.
4. Once a HTTP request is sent by a client the switch receives the request and is directed to the POX controllers by their respective ports.
5. Random load balancing algorithm is running on both controllers.
6. Once both The POX controller and load balancer gets ready.
7. Ten (10) to seventy (70) concurrent requests are respectively generated by the Siege load testing tool.
8. Concurrent requests are distributed to both controllers for sharing a load.
9. Then HTTP responses are sent back to the clients.
10. The result will be evaluated in terms of three network parameters (Response time (sec), Transaction rate (trans/sec), Throughput (KB/sec))

4.2. Proposed Load Balancing Algorithm

The proposed Load Balancing algorithm in this paper is evaluated and compared with round robin, weighted round-robin and least connections which are one of the most cited algorithms in this field. The results are compared in terms of the three network performance parameters Response time, transaction rate and throughput under various amounts of loads for each algorithm. This paper utilized an Analysis of load balancing algorithms based on different topologies by increasing the number of servers and propose a Random load balancing algorithm in which it randomly distributes the workload on available resources and provide less response time, high transaction rate and throughput compared to other load balancing algorithms in case of tested topologies in this work.

4.2.1. The workflow for the proposed Random Load Balancing Algorithm

Researcher in [37] shows, the Analysis of average load on a server when a random picking of a server is used.

- Let there be k requests (or jobs) J1, J2, … Jk
- Let there be n servers be S1, S2 … Sk.
- Let time taken by i'th job be Ti
- Let Rij be loaded on server Si from Job Jj.
- Rij is Tj if j'th job (or Jj) is assigned to Si, otherwise 0. Therefore, value of Rij is Tj with probability 1/n and value is 0 with probability (1-1/n)
- Let Ri be load on i'th server

Average Load on i'th server 'Ex (R_i)'

Applying Linearity of Expectation in [38],

=

$$\sum_{j=1}^{k} Ex\left[R_{ij}\right]$$

=

$$\sum_{J=1}^{K} T_i/n$$

= (Total Load)/n

4.3. Performance Evaluation Metrics for the Proposed Topology

In this work, the following parameters are going to be used to measure the performance of the Proposed Multi-Controller Topology.

- **Response time**

Response time is the time that algorithms takes to respond to a given client request. This includes the sum of total waiting time, transmission time, and service time that the system requires. Therefore, minimizing response time will be the goal to optimize performance and efficiency of Load balancing Algorithms.

- **Throughput**

Throughput is the metrics fir measuring the successfully accomplished work, and in the case of the load balancer, throughput is the total number of successfully accomplished per second.

It's an insightful metric to measure since higher throughput indicates higher efficiency of our load balancers, signalling healthy load balancing.

- **Transaction rate**

Transaction rate is a measurement of transactions performed per second.in the case of load balancing algorithms, it depends on the response time of an algorithm, if the response time is less, and the transaction performed per second will be high. Therefore, maximizing the number of transactions performed per second will be the goal to optimize the performance of load balancing algorithms.

CHAPTER FIVE

5. PERFORMANCE EVALUATION

In this Chapter, SDN based load balancing algorithms performance is evaluated using the Mininet SDN network simulation tool which is very widely employed in SDN research and POX controller. Four (4) load balancing algorithms are compared with each other, Random, Round-robin, Weighted Round-robin and least load balancing algorithms, in terms of performance evaluation metrics, response time/sec, transaction rate trans/sec and throughput (MB/sec)by increasing the number of concurrent client requests. The other issue discussed in this section is, evaluating and comparing the performance of load balancing algorithms by increasing the number of servers and controllers based on three different topologies and coming up with a topology having a better load balancing performance among tested topologies.

5.1. Simulation Setup

Mininet 2.30 is used to emulate the SDN network on a Desktop with Intel(R) Core (TM) i3-3220 CPU@3.30GHZ,10.0GB RAM which has 64-bit operating system and to run Ubuntu 16.04 Linux operating system Oracle virtual box 6.0.24 is used as a hypervisor. SDN POX controller is used as a remote controller in which the proposed algorithm is deployed. The proposed multi-controller is connected to the Open Flow switch on port 6633 and 6630 to generate ten to seventy concurrent HTTP requests and to test load the SIEGE tool is used.

In this work, there are three topologies tested with different scenarios.

- **Topology 1:** in topology one three (3) clients and three(3) servers are connected to a single Open Switch and a single POX controller running different load balancing algorithms.

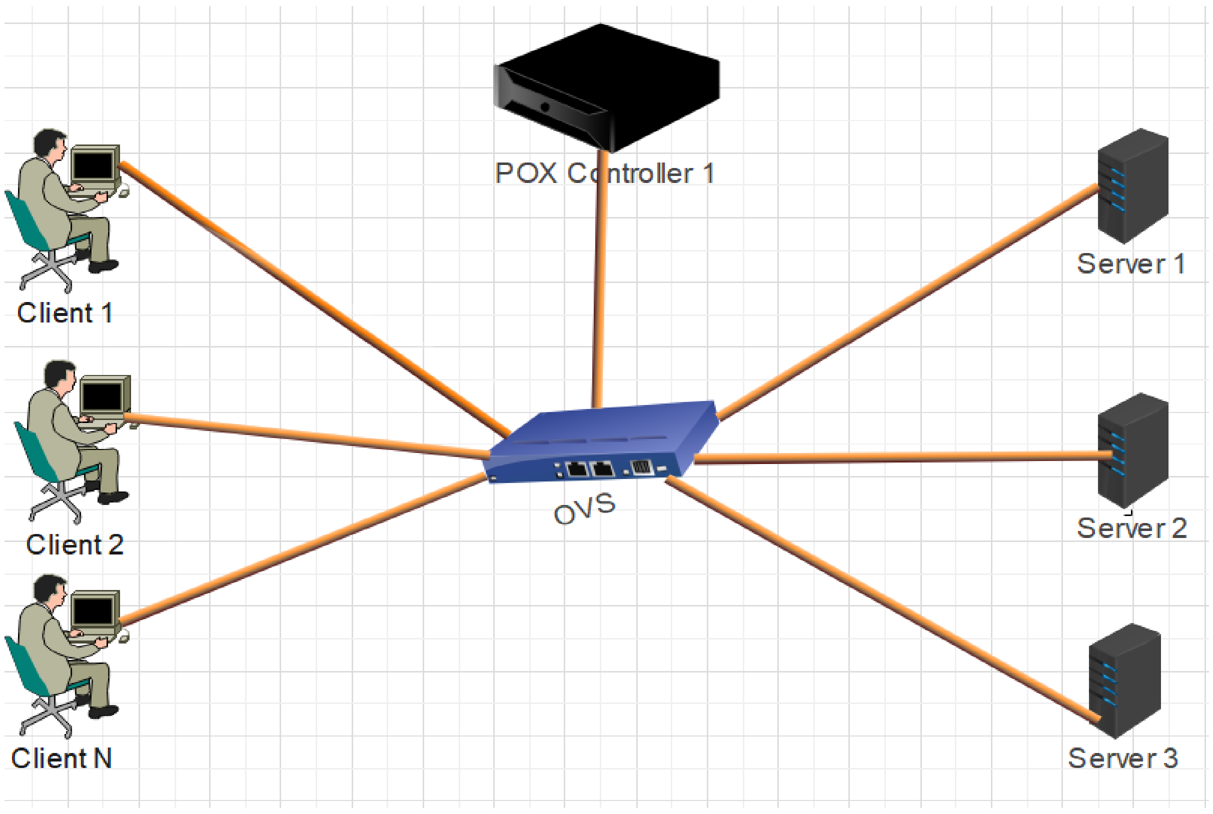


Fig 5-1 Topology 1

- **Topology 2: In topology 2**, three (3) clients and six (6) servers connected to a single Open Flow switch and a single POX controller running different load balancing algorithms. The number of servers is increased in order to test whether the response time is minimized compared to topology one or not.

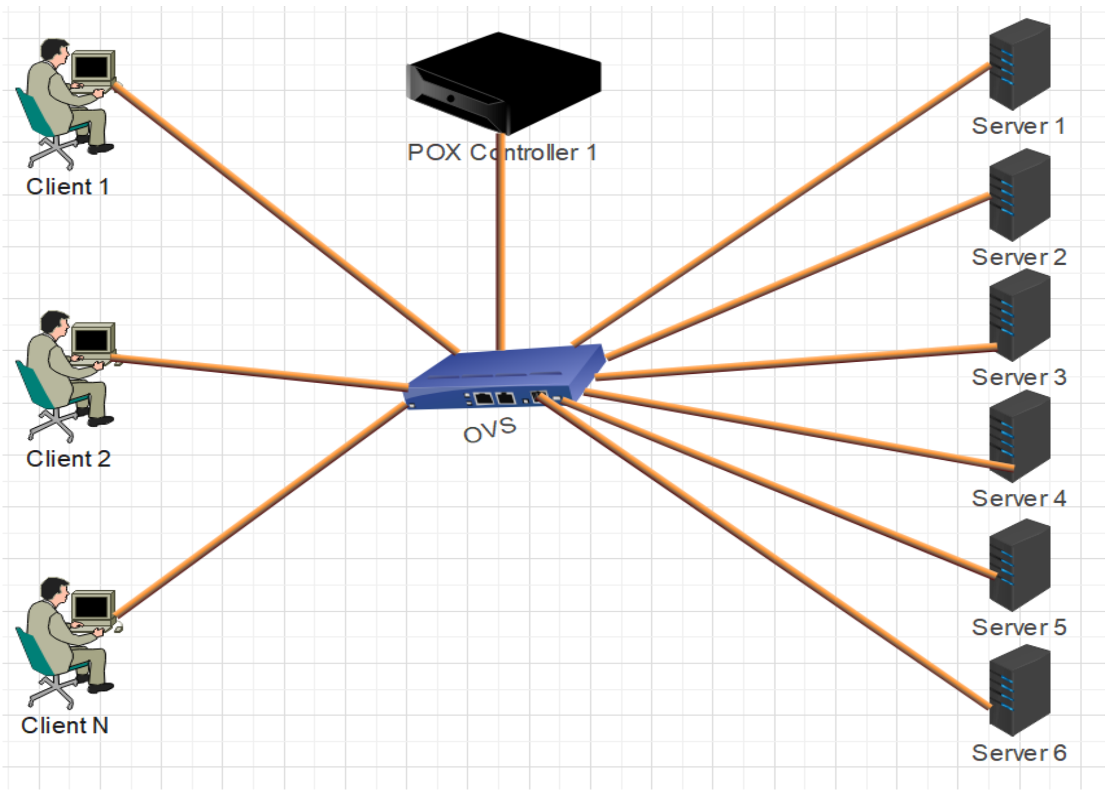


Fig 5-2 Topology 2

- **Topology 3 (The proposed Topology):** the last topology is topology 3. In this topology three (3) clients and six (6) servers are connected to a single Open Flow switch which is connected to a multiple controller by port 6633 and 6630 using an open flow model and each controller is running a Random load balancing algorithm. Here in this topology, multiple controllers are used to share a load from each other. A request sent from clients is distributed to both controllers running their own load balancing algorithm, which improves the performance of load balancers.

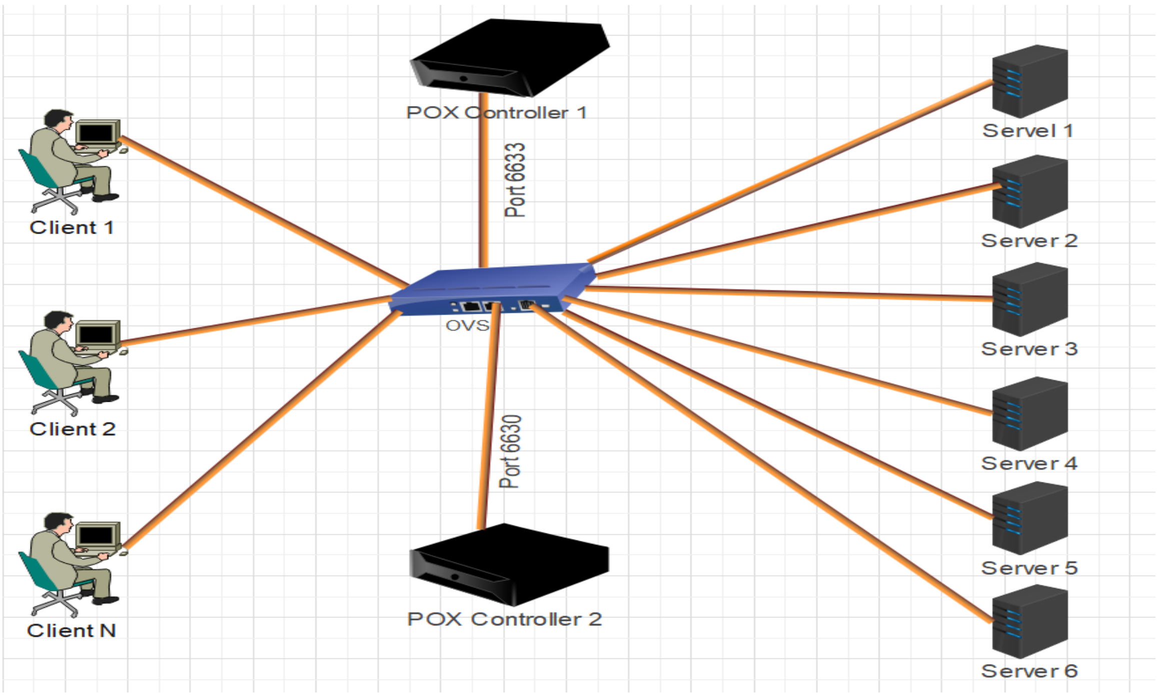


Fig 5-3 Topology 3

5.2. Results and Observations

The result for the performance analysis of the four load balancing algorithms in terms of different topologies and the result for the proposed topology and the comparisons are presented in the form of both tabular and graphical for simple understanding.

5.3. Results and Observations for Topology 1

5.3.1. Response time (sec) for topology 1

The response time (sec) of random, round robin, weighted round robin and least load balancing algorithms for topology one is presented in table 5-1 below. By increasing concurrent HTTP requests of clients from ten (10) to seventy (70), respectively.

Table 5-1 Topology 1 Response Time

Concurrent Users	Load Balancing Algorithms			
	Response time(sec)			
	Random	Round Robin	Weighted Round Robin	Least Load
10	0.11	0.11	0.18	0.15
20	0.20	0.22	0.32	0.28
30	0.30	0.30	0.39	0.37
40	0.40	0.39	0.45	0.39
50	0.46	0.42	0.55	0.45
60	0.55	0.45	0.77	0.50
70	0.58	0.53	0.89	0.45

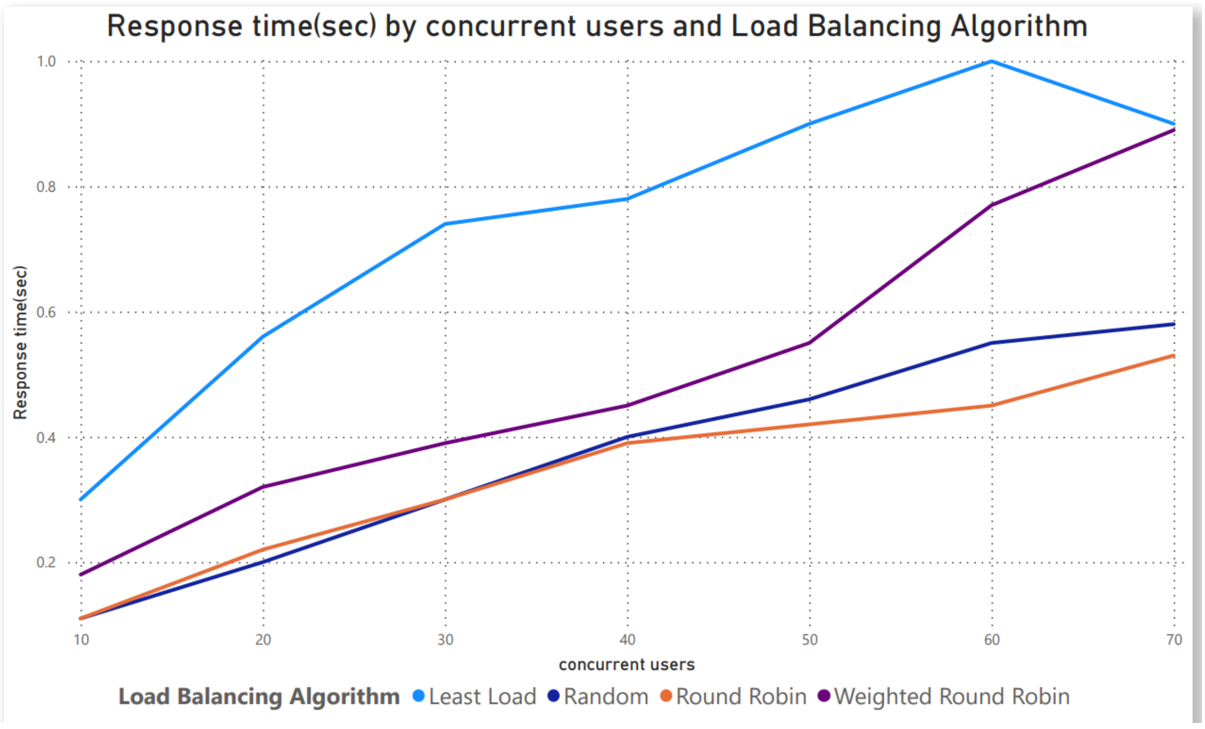


Fig 5-4 Response time graph

- The response time (sec) graph in figure 5.4 shows that, the response time (sec) of random round robin, weighted round robin and least load balancing algorithms increases as the number of concurrent users increases, but the response time (sec) of random and round robin load balancing algorithm is less and close when compared to weighted round robin and least load balancing algorithm. This is because the random and round robin load balancing algorithm does not check for any requirement. The random simply assigns a request for a server and the round robin simply assigns requests in a predefined pattern in cyclic fashion.

- However, the response time for both weighted and least load balancing algorithms is increasing linearly as the concurrent request is increasing because they booths check for a requirement were as, weighted load balancing algorithm check for weight of each server before assigning the request and the least load balancing algorithm checks for the minimum server load before assigning a request for a server.

5.3.2. Transaction rate (trans/sec)

The transaction rate in (trans/sec) for a topology 1 of random. round robin, weighted round robin and least load balancing algorithms have been presented by increasing the number of concurrent users starting from ten(10) to seventy (70), respectively.

Table 5-2 Topology 1 transaction rate

Concurrent Users	Load Balancing Algorithms			
	Transaction Rate(trans/sec)			
	Random	Round Robin	Weighted Round Robin	Least Load
10	89.03	93.81	55.06	68.03
20	96.94	90.16	59.83	69.66
30	97.97	98.03	59.03	68.94
40	97.51	98.24	58.58	70.65
50	99.95	97.19	58.99	71.75
60	96.70	100.88	59.35	68.38
70	99.32	99.10	58.34	68.01

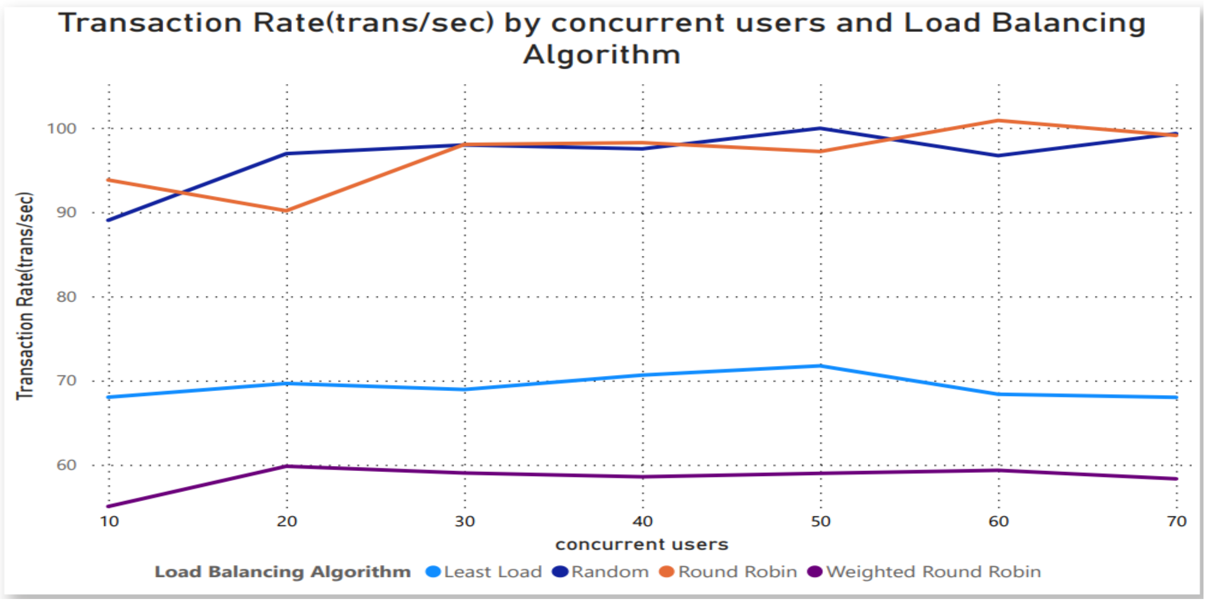


Fig 5-5 Topology 1 Transaction rate

- The transaction/sec for the topology one shows that random and round robin load balancing algorithms have a better transaction rate (trans/sec) as the number of concurrent users increases when compared to weighted round robin and least load server cluster load balancing algorithms. This is because they have less response time (sec) and handle more transaction/sec compared to that of weighted round robin and least load balancing algorithms as shown in figure 5.4.

5.3.3. Throughput (MB/sec) for topology 1

The throughput (MB/sec)for topology_1 of random, round robin, weighted round robin and least load balancing algorithms have been presented in table 5-3 below by increasing the number of concurrent users from ten(10) to seventy(70), respectively.

Table 5-3 Topology 1 throughput

Concurrent Users	Load Balancing Algorithms			
	Throughput (MB/sec)			
	Random	Round Robin	Weighted Round Robin	Least Load
10	0.13	0.13	0.07	0.09
20	0.13	0.12	0.08	0.09
30	0.13	0.13	0.08	0.09
40	0.13	0.13	0.08	0.09
50	0.13	0.12	0.08	0.09
60	0.12	0.13	0.08	0.09
70	0.13	0.13	0.07	0.09

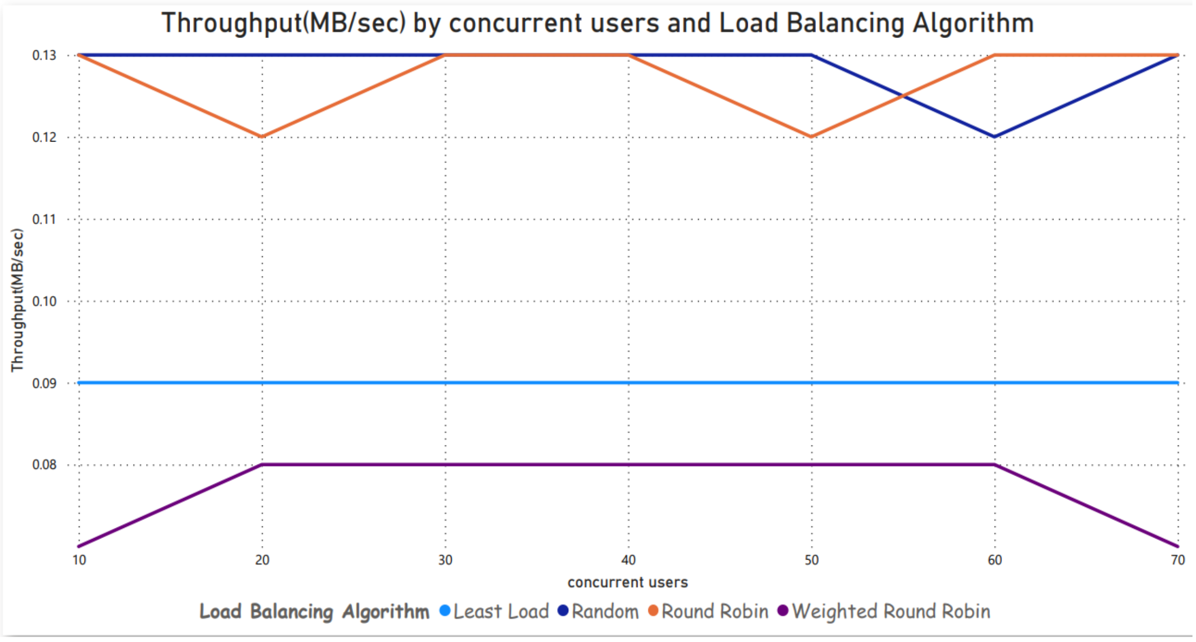


Fig 5-6 Topology 1 Throughput

- The result observed in topology_1 the throughput (MB/sec) of random and round robin load balancing algorithm has shown a highest result compared to that of Weighted and least load balancing algorithm. And, the reason for this is that both random and round robin load balancing algorithms have a better response time relatively and this will achieve a higher throughput than weighted and least load balancing algorithms as shown on figure 5.6.

5.4. Results and Observations for Topology 2

5.4.1. Response time for topology 2

The response time/sec of random, round robin, weighted round robin and least load balancing algorithms for topology two (2) is presented in table 5-4 below respectively. By increasing concurrent HTTP requests of clients from ten (10) to seventy (70).

Table 5-4 Topology 2 Response time

Concurrent Users	Load Balancing Algorithms			
	Response time (sec)			
	Random	Round Robin	Weighted Round Robin	Least Load
10	0.13	0.12	0.29	0.15
20	0.28	0.25	0.58	0.35
30	0.41	0.41	0.68	0.58
40	0.59	0.51	0.98	0.38
50	0.87	0.60	0.84	0.41
60	0.81	0.85	1.49	0.52
70	0.82	0.87	1.53	0.54

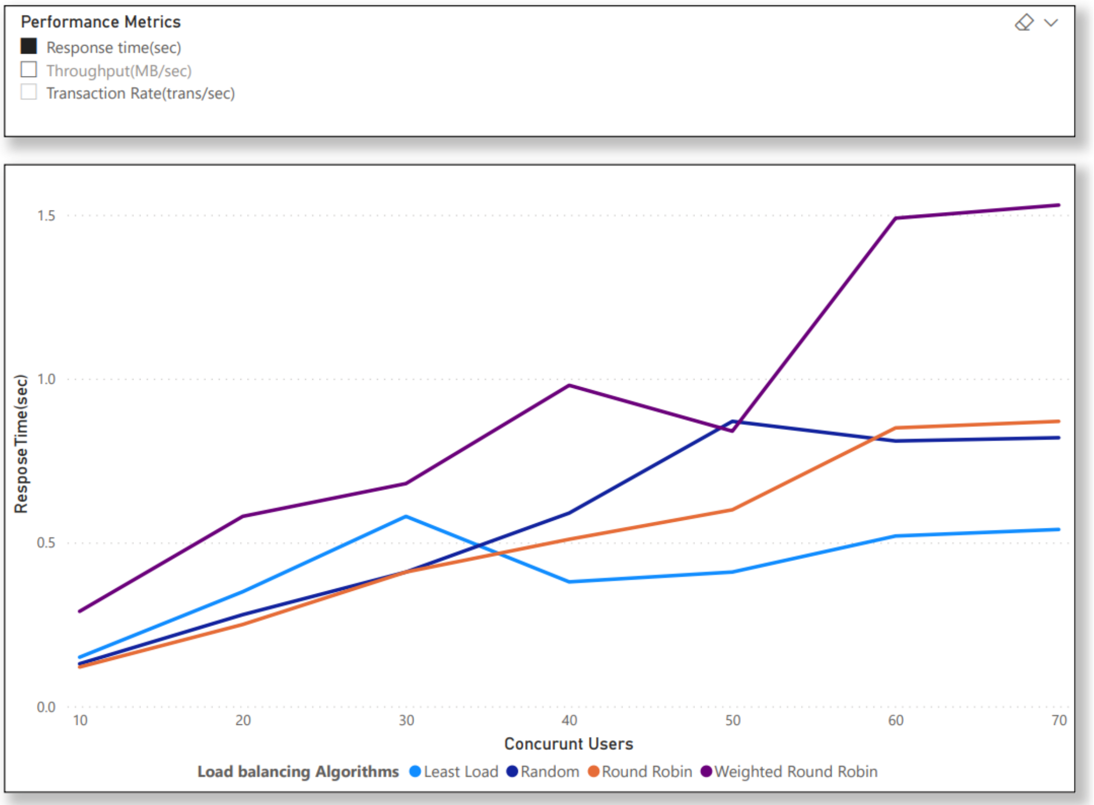


Fig 5-7 Response Time for Topology 2

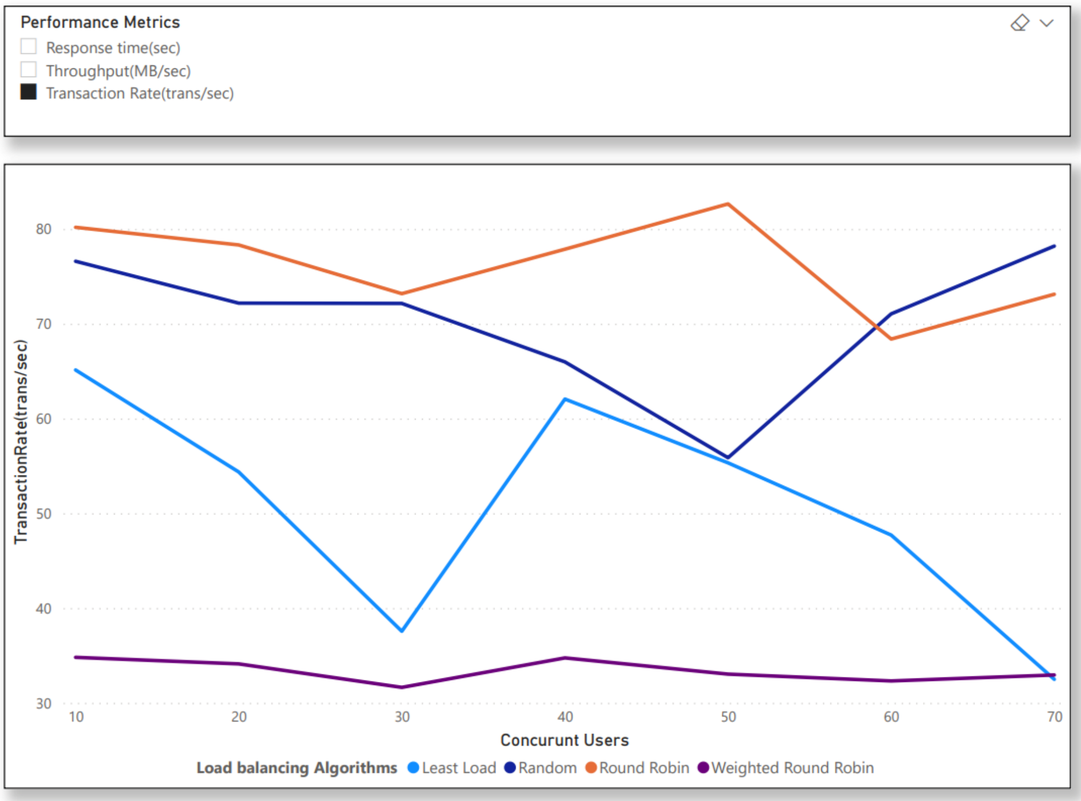
- The response time (sec) graph in 5.7 shows that the response time (sec) of random and round robin load balancing algorithms have less and close response time compared to weighted round robin and least load balancing algorithms.
- However, the response time for Topology _1 is less in the case of random and round robin load balancing algorithms compared to the response time/sec for topology 2. But the response time/sec for Topology_2 is less when compared to topology_1 in the case of weighted round robin algorithm and least load balancing algorithm. The reason for this is weighted round robin checks for the weight that each server can handle before sending the request and the least load balancing algorithm checks the load for each server before sending the request.

5.4.2. Transaction rate (trans/sec) for Topology 2

The transaction rate in (trans/sec) of random, round robin, weighted round robin and least load balancing algorithms for Topology 2 have been presented I table 5-5 below by increasing the number of concurrent HTTP requests from ten(10) to seventy(70), respectively.

Table 5-5 Topology 2 transaction

Concurrent Users	Load Balancing Algorithms			
	Transaction Rate(trans/sec)			
	Random	Round Robin	Weighted Round Robin	Least Load
10	76.60	80.18	34.81	65.13
20	72.18	78.32	34.12	54.37
30	72.15	73.18	31.64	37.56
40	65.98	77.87	34.75	62.06
50	55.87	82.65	33.04	55.33
60	71.05	68.39	32.32	47.71
70	78.20	73.12	32.95	32.49



Performance Metrics

☐ Response time(sec)
☐ Throughput(MB/sec)
■ Transaction Rate(trans/sec)

Fig 5-8 Topology 2 Transaction rate

- The transaction(sec) graph in figure 5.8 shows that the transaction rate of random and round robin load balancing algorithms are higher compared to weighted round robin and least load balancing algorithms
- When comparing the transaction/sec of topology_1 with topology_2 the transaction /sec for topology_1 is better.
- Even though the response time for weighted round robin and least load balancing algorithm were less in topology_2, the transaction /sec they handle is less compared to topology_1.

5.4.3. Throughput for topology 2

The throughput in (MB/sec)of round robin, weighted round robin and least load server cluster load balancing algorithms by increasing the number of concurrent users starting from 10 concurrent user to 70 concurrent users with gradual increase of 10 concurrent users in between has described in table 5-6 below:

Table 5-6 Topology 2 throughput (MB/sec)

Concurrent Users	Load Balancing Algorithms			
	Throughput(MB/sec)			
	Random	Round Robin	Weighted Round Robin	Least Load
10	0.12	0.11	0.09	0.10
20	0.12	0.11	0.07	0.08
30	0.11	0.11	0.07	0.06
40	0.10	0.12	0.06	0.09
50	0.13	0.12	0.06	0.08
60	0.11	0.10	0.05	0.07
70	0.12	0.11	0.05	0.05

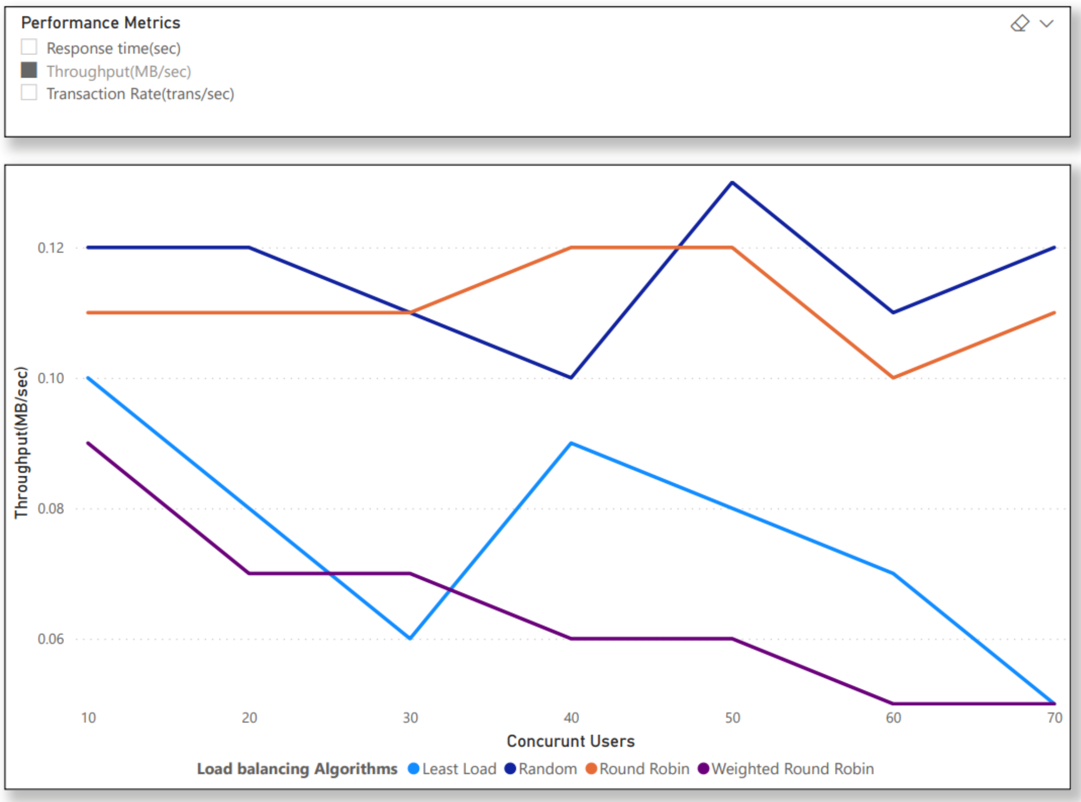


Fig 5-9 Throughput for topology 2

- According to the result observed the throughput (MB/sec) of random and round robin load balancing algorithm has shown a highest result compared to that of Weighted and least load balancing algorithm. And the reason for this is that both random and round robin load balancing algorithms have a better response time relatively and this will achieve a higher throughput than weighted and least load balancing algorithms as shown on figure 5.9.
- Comparing the throughput of load balancing algorithms based on topology, the throughput for topology_1 is higher than the throughput for topology_2. This is achieved because the response time for topology_1 is less compared to topology_2.

5.5. Results and Observations for the proposed multi-controller Topology (Topology 3)

According to the performance analysis of random, round robin, weighted round robin and least load balancing algorithms from the above figure 5-1, figure 5-2 and figure 5-3 topologies, Random load balancing algorithm is selected as having a better Response time (sec) transaction rate (trans/sec) and throughput (MB/sec) compared to round robin weighted round robin and least load balancing algorithms. Therefore, a random load balancing algorithm is used for further analysis in the proposed Open Flow based multi-controller topology.

- The proposed topology is a multi-controller topology which can only be achieved using the Open Flow model.
- The proposed topology is a new topology in which a multiple controller is connected on a single open switch through predefined port 6633 and 6630.
- In the proposed topology the concurrent requests sent by the client are shared on the two controllers since both controllers are running a random load balancing algorithm, the response time, transaction rate and throughput for a load balancer will be improved.

let’s assume that a client is sending 30 concurrent HTTP requests, So here the request received will not go to only one controller the request will be shared for both controllers and the response received will be fast, high transaction rate and high throughput.

5.5.1. Response time for proposed topology (topology 3)

The response time/sec of the random load balancing algorithm for the proposed multi-controller topology is compared with topology_2 (a topology with a single controller) and presented in table 5-7 below, by increasing concurrent HTTP requests of clients from ten (10) to seventy (70) respectively.

Table 5-7 Proposed Topology Response Time/sec

Concurrent Users	Random Load balancing Algorithm	
	Response time(sec)	
	Single Controller	Multi-Controller
10	0.13	0.11
20	0.28	0.20
30	0.41	0.30
40	0.59	0.40
50	0.87	0.46
60	0.81	0.55

70	0.82	0.58

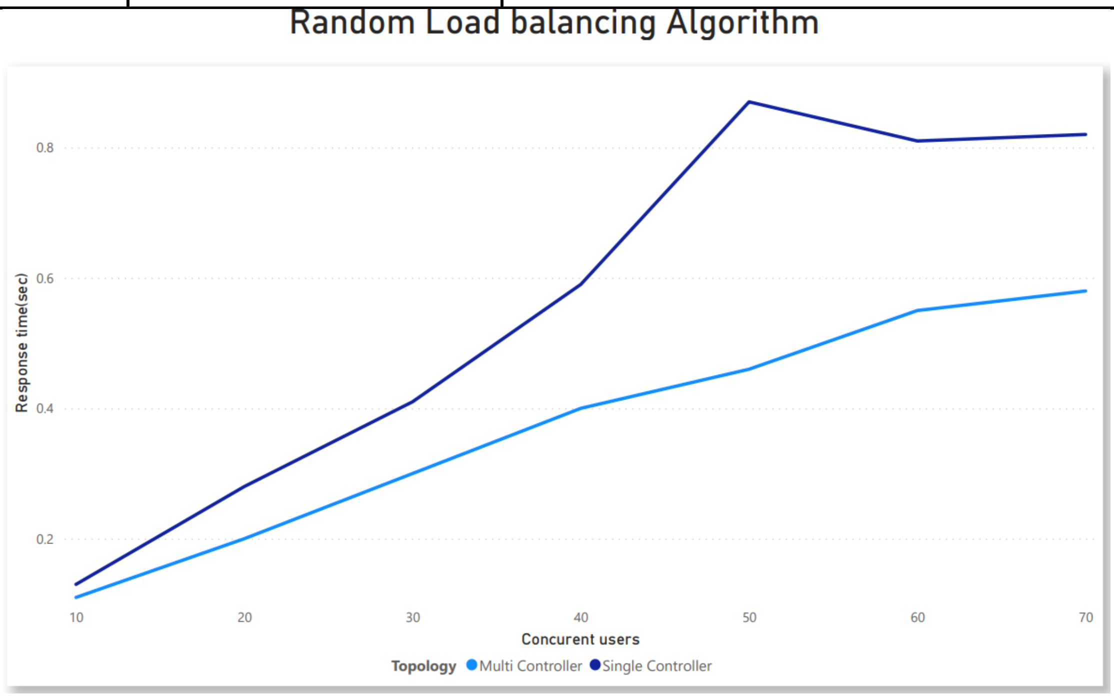


Fig 5-10 Topology 3 Response time

As the result is observed, the response time for the proposed multi-controller topology is less when compared to a topology with a single controller.

- This can be achieved because of having multiple controllers and sharing the load to both controllers.

5.5.2. Transaction rate for the proposed topology (topology 3)

The transaction /sec of the random load balancing algorithm for the proposed multi-controller topology is compared with topology_2 (a topology with a single controller) and presented in table 5-8 below, by increasing concurrent HTTP requests of clients from ten (10) to seventy (70), respectively.

Table 5-8 proposed topology transaction rate

Concurrent Users	Random Load balancing Algorithm	
	Transaction Rate(trans/sec)	
	Single Controller	Multi-Controller
10	76.60	89.03
20	72.18	96.94
30	72.15	97.97
40	65.98	97.51
50	55.87	99.95
60	71.05	96.70
70	78.20	99.32

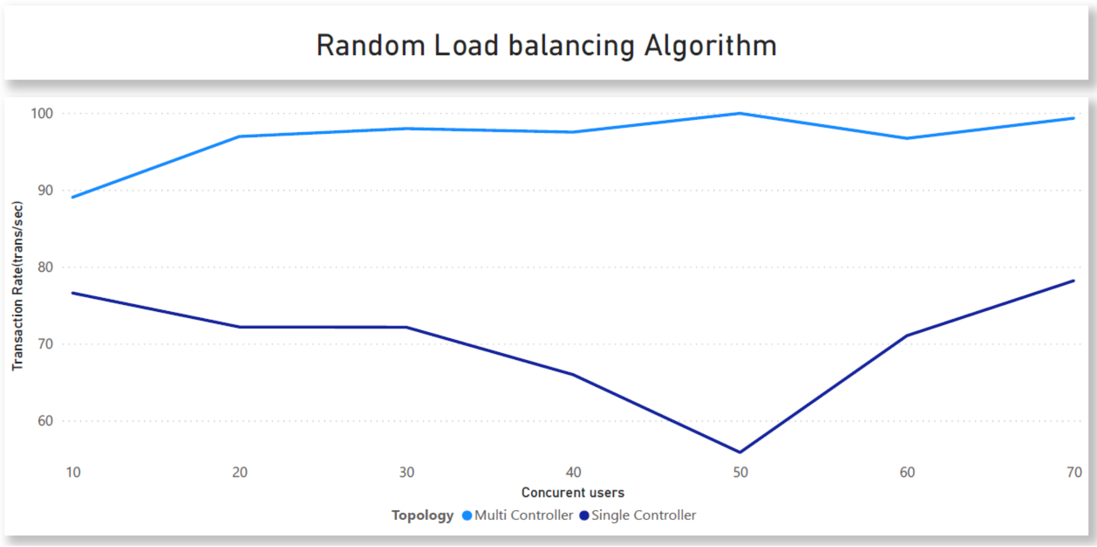


Fig 5-11 Transaction rate for topology 3

- As it is observed from the graph, the transaction rate for the proposed multi-controller topology is higher when compared to a topology with a single controller.
- The transaction rate is linearly increasing as the concurrent request increases, this is because the response time for the multi-controller topology is increasing linearly as the number of concurrent requests increase.

5.5.3. Throughput for the proposed topology (topology 3)

The throughput (MB/sec) of the random load balancing algorithm for the proposed multi-controller topology is compared with topology_2 (a topology with a single controller) and presented in table 5-9 below, by increasing concurrent HTTP requests of clients from ten (10) to seventy
(70), Respectively.

Table 5-9 proposed topology throughput

Concurrent Users	Random Load balancing Algorithm	
	Throughput (KB/sec)	
	Single Controller	Multi-Controller
10	120	140
20	120	130
30	110	130
40	100	130
50	100	130
60	130	130
70	110	120

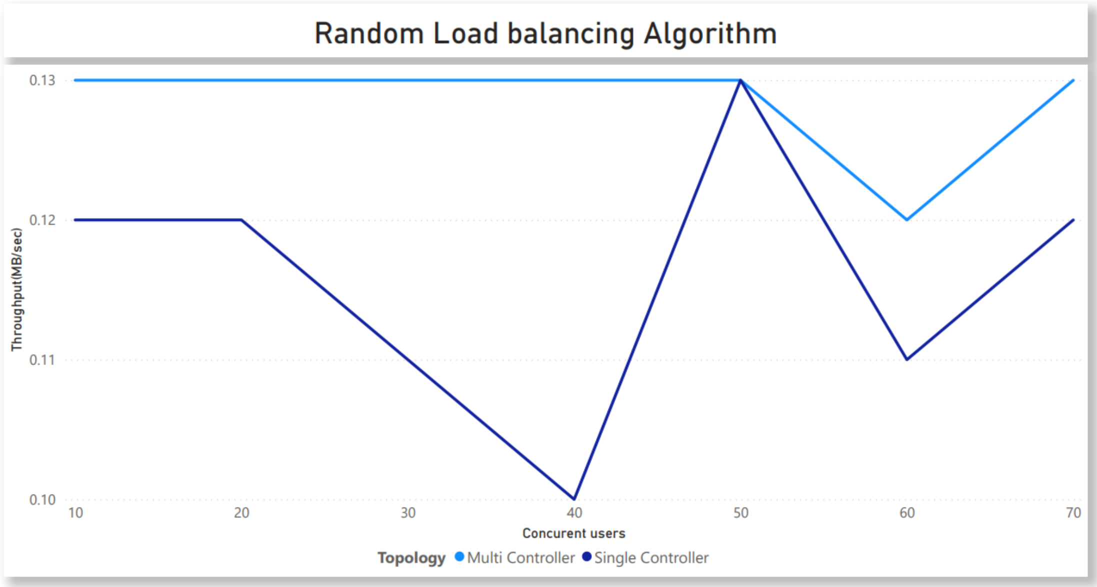


Fig 5-12 Topology 3 Throughput

- As the result is observed, the throughput for the proposed multi-controller topology is higher when compared to a topology with a single controller.

- This can be achieved because of having multiple controllers and loads are shared to both controllers, the number of requests successfully completed is higher.

5.6. Comparative Analysis of Results

The comparative analysis of four topologies in terms of a response time (sec) by using Open Flow model is presented in table 5-10 below in a tabular form. As we can see from the table, the response time, transaction rate and throughput value for a proposed Open Flow based Multi-Controller topology is better than topologies presented by researcher [36] and researcher [13]. In this section, response time metrics is used to compare the four topologies because of response time is their common performance metrics for both researchers, and the transaction rate and throughput is compared with topology one of researcher [13].

Table 5-10 Comparative analysis for Response time

Number of requests	Researcher in [36]Topology Two	Researcher in [36]Topology Three	Researcher in [13] Topology	Single Controller Topology	Proposed Multi-Controller Topology
10	0.397	0.44	0.14	0.13	0.11
20	0.41	0.49	0.32	0.28	0.20
30	0.44	0.55	0.45	0.41	0.30
40	0.47	0.59	0.61	0.59	0.40
50	0.0.51	0.65	0.77	0.87	0.46
60	0.558	0.71	0.92	0.81	0.55
70	0.59	0.78	1.1	0.82	0.58

Table 5-1 1 Comparative analysis for Transaction rate

Number of requests	Researcher in [13] Topology	Single Controller Topology	Proposed Multi-Controller Topology
10	85.63	76.60	89.03
20	82.37	72.18	96.94
30	91.42	72.15	97.97
40	69.10	65.98	97.51
50	94.18	55.87	99.95
60	75.71	71.05	96.70
70	67.72	78.20	99.32

Table 5-1 2 Comparative analysis for Throughput KB/sec

Number of requests	Researcher in [13] Topology	Single Controller Topology	Proposed Multi-Controller Topology
10	130	120	140
20	130	120	130
30	110	110	130
40	100	100	130
50	110	100	130
60	100	130	130
70	100	110	120

CHAPTER SIX

6. CONCLUSIONS, CONTRIBUTIONS AND FUTURE WORKS

6.1. Conclusions

In this work, performance analysis is done on four SDN based load balancing algorithms namely random, round robin, weighted round robin and least load balancing algorithm in terms of three network performance metrics response time/sec, transaction rate/sec and throughput (MB/sec)based on topology 1 and topology 2 by increasing the number of concurrent users from ten(10) to seventy(70) which is simulated using SDN Open Flow model in open source POX controller and Open Flow switches in mininet for topology creation. According to the result, a random load balancing algorithm is selected as a better load balancing algorithm in terms of network performance metrics response time/sec, transaction rate/sec and throughput (MB/sec).

In addition, a new Open Flow model based multi-controller topology is proposed and compared with topology 2 which has a single controller using random load balancing algorithm for improving the performance of SDN based load balancing algorithms. and the result obtained shows that a proposed multi-controller topology has improved the performance of SDN based load balancing in terms of network performance metrics response time/sec, transaction rate /sec and throughput (KB/sec).

6.2. Contributions

- Propose a new Open Flow based multi-controller topology for performance improvement of SDN based load balancing algorithm.
- Performance evaluation of random, Round robin, weighted round robin and least load balancing algorithm based on different topology is done. And the proposed topology improved the response time (sec) by reducing an average of 30.12%, increasing the transaction rate (trans/sec) by an average of 39.44% and also increasing the throughput (KB/sec) by an average of 10.56% when compared with a single controller topology using random load balancing algorithms in SDN POX controller.

- Since SDN deployment is at initial stage in Ethiopia, this study provides an insight that Open Flow based multi-controller topology can be implemented with different load balancing algorithms to achieve a better operational efficiency.

6.3. Future Works

To have a better performance of load balancing algorithms in the SDN network the following future work might be important.

- In this study, the proposed Open Flow based multi-controller topology is done in a simulation environment using POX Controller. As a future work, the proposed topology can be tested on a real SDN network.
- And since there are different controllers supporting different Open Flow protocol versions the proposed Open Flow multi-controller topology can be tested on different Open Flow protocol versions.
- Since there are different network performance evaluation metrics like Latency, Error rate, Reliability and Fault tolerance, SDN load balancing algorithms can be evaluated based on those metrics as a future work.

REFERENCES

[1] A. Hakiri, A. Gokhale, P. Berthou, D. C Schmidt, T. Gayraud, et al., “Software-defined networking:Challenges and research opportunities for future Internet,” *Computer Networks*, vol. 75, pp. 453–471, 2014.

[2] M. Erel, Z. Arslan, Y. Ozcevik, B. Canberk, et al., “Software-defined wireless network (SDWN) a new paradigm for next generation network management. In Modeling and Simulation of Computer Networks and Systems,” *Elsevier: Amsterdam, the Netherlands*,” vol. 17, pp. 751–766, 2015.

[3] A. Abdelaziz, A. Fong, A. Gani, G. Usman, K. Suleman, A. Adnan khunzada, Hamid Talebian, Kim-Kwang Raymond Choo, et al., “Distributed controller clustering in software defined networks,” 6 April 2017.

[4] A. Neghabi, N. Navimipour, M. Hosseinzadeh, A. Rezaee, *et al., “Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature,*” IEEE Access [Online] Available: 2018, 6, 14159–14178.

[5] P. Martinez-Julia, A. Skarmeta. “*Empowering the internet of things with software defined networking,” In White Paper, IoT6-FP7 European Research Project; 2014*. [Online]. Available: https://www.semanticscholar.org/search?q=Empowering-the-Internet-of-Things-with Software &sort=relevance (accessed on 3 May 2020).

[6] D. Kreutz, F. Ramos, P. Verissimo, C. Rothenberg, S. Azodolmolky, S. Uhlig, et al., “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE 103, 1* (2015), 14–76.

[7] I. Akyildiz, A. Lee, P. Wang, M. Luo, W Chou, et al., “A roadmap for traffic engineering in SDN-Open Flow networks,” *Computer Networks,* Vol. 71 pp. 1–30, 2014.

[8] S. Thabo, M. Thabiso, A. Stephen , K. Kefalotse, D. Setso, B. Gabanthone, S. Seth et al., (2020), “*Intelligent Load Balancing Techniques in Software Defined Networks,”: survey, Electronics* , *9* (7), 1091; [ONLINE] Available: https://doi.org/10.3390/electronics9071091

[9] S. Cisco, “SDN Architecture,” [Online]. Available:

https://www.cisco.com/c/en/us/solutions/software-defined networking/overview.html

[10] W. Raniyah, A. Rami A. Suheib, 2021 “SDN-Open Flow Topology Discovery: An Overview of Performance Issues,” *Applied Science*., 11, 6999: https://doi.org/10.3390/app11156999

[11] R. Mohammad, M. Shahrulniza, A. Muhammad, M. Mazliham, “*A Systematic Review of Load Balancing Techniques in Software-Defined Networking,”* [Online] Available: 10.1109/ACCESS.2020.2995849

[12] S. Sabiya, "Weighted round-robin load balancing using software defined networking," *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 6, pp. 621-625, 201.

[13] Wubishet Abebe, “Performance Evaluation of Server Cluster Load balancing Algorithms Using SDN Open Flow Model,” Addis Ababa, Ethiopia, 2016.

[14] M. Priyadarsini, J. Mukherjee, P. Bera, S. Kumar, A. Jakaria, M. Rahman, et al., “An adaptive load balancing scheme for software-defined network controllers,” *Computer Network*, vol. 7, pp. 23, 2019

[15] Alexis de Talhouët, “The evolution of software defined networking,” June 17, 2021.

[16] K. Karamjeet, S. Japinder and S. Navtej, “Network Programmability Using POX Controller,” *Department of Computer Science and Engineering*, 2015.

[17] Haeeder Munther Noman and Mahdi Nsaif Jasim, “POX Controller and Open Flow Performance Evaluation in Software Defined Networks (SDN) Using Mininet Emulator,” 2020 IOP Conf. Ser.: Mater. Sci. Eng. 881 012102.

[18] A. Tecmint, “Load Testing Web Servers with Siege Benchmarking Tool,” [Online]. Available: https://www.tecmint.com/load-testing-web-servers-with-siege-benchmarking-tool/

[19] F. Carlos, L. Jose, Z. Muñoz “Software Defined Networking (SDN) with Open Flow1.3, Open vSwitch and Ryu,” UPC Telematics Department, 20 July 2015.

[20] M. Ali, “Inside-Open Flow,” [Online]. Available: https://confignetworks.com/inside-Open Flow/

[21] G. Paul, B. Chuck, “Software Defined Networks a Comprehensive Approach 1st edition,” Harlow: Prentice Hall, 2014.

[22] M. Deep, R. Karthik, “Network Routing (Second Edition),”Interconnection Networks, 2018.

[23] Oswald Coker, Siamak Azodolmolky, “Software Defined Networking with Open Flow,” October 2017.

[24] Khan, M. Ali, N. Sher, Y. Asim, W. Naeem, and M. Kamran, “Software-Defined Networks (SDNs) and Internet of Things (IoTs): A Qualitative Prediction for 2020,” *International Journal of Advanced Computer Science Application,* Vol. 7, No. 11, pp. 385–404, 2016

[25] N. Hamid, S. Rasool, M. Sayed, I. Faghih, “Load balancing in software defined networking using controller placement,” Department of Computer Engineering [Online] Available at: r.sadeghi@iauda.ac.ir) , 2020.

[26] B. Sumit, S. Japinder, B. Shaheed, “Introduction *to Load balancing and Strategies used in software defined networking,”* Ferozepur, Punjab: 2017.

[27] C. Connor, “What *is Open Flow,*” [Online]. Available: https://www.sdxcentral.com/networking/sdn/definitions/what-is-Open Flow/: November 2020.

[28] L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, "Ofswitch13: Enhancing ns-3 with Open Flow 1.3 support," in Proceedings of the Workshop on vol 3, pp. 33-40, 2016.

[29] IBM, “*What Is Software defined networking,*” [Online]. Available: https://www.ibm.com/cloud/blog/software-defined-networking : 15 April 2021

[30] G. Paul, B. Chuck and C. Timothy, “*Software Defined Networks,” 2nd edition,* 2017.

[31] A. Jehad li, L. Seungwoon, R. Byeong-hee, “*Performance Analysis of POX and Ryu with Different SDN Topologies,”* In ICISS '18: Proceedings of the 2018 International Conference on Information Science and System, April 2018, pp.244-249

[32] S. Nazari, P. Gerla, M. Hoffmann, C. Kim, A. Capone, et al., “*Software defined naval network for satellite communications (SDN-sat),*” MILCOM: 2016.

[33] S. Fizi, and S. Askar, "A novel load balancing algorithm for software defined network based datacenters,” Proceedings. *IEEE International. Conference Broadband*

Community. Next Generat. Network Multimedia Application. (CoBCom), vol. 7, pp. 1-6, Sep. 2016.

[34] M. Omran, Z. Al Saheli, A. Zainal, N. Zakaria, Z. Abal, et al., " Software Defined *Network based Load Balancing for Network Performance Evaluation*," vol.13, No 4, 2022.

[35] L. Chen, M. Qiu, J. Xiong, "An SDN-Based fabric for flexible data-center networks" in Cyber Security and Cloud Computing (CSCloud)," IEEE 2nd *International Conference*, pp. 121-126, 2015.

[36] Venkatesh Kodela, "Improving load balancing mechanisms of software defined networks using Open Flow," Jawaharlal Nehru Technological University, India, August 2016.

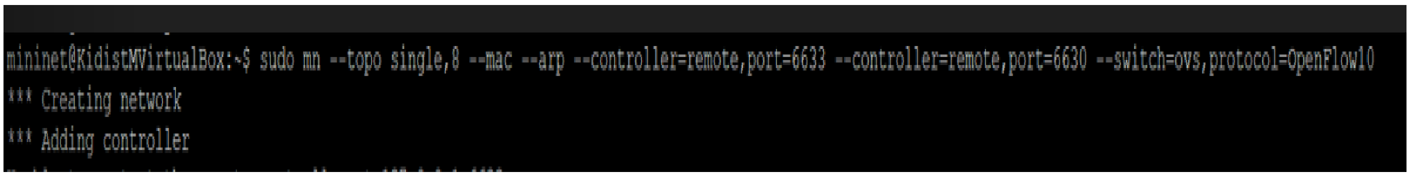
[37] GeeksforGeeks, "Load Balancing on Servers (Randomized Algorithm)," [Online]. Available: https://www.geeksforgeeks.org/load-balancing-on-servers-random-algorithm/:15 Nov, 2015.

[38] GeeksforGeeks, "Linearity of expectations," [Online]. Available: https://www.geeksforgeeks.org/linearity-of-expectation/: 28 June, 2021.

APPENDICES

A. Appendix A: Simulation in mininet

A.1. Creating Multi-Controller Topology

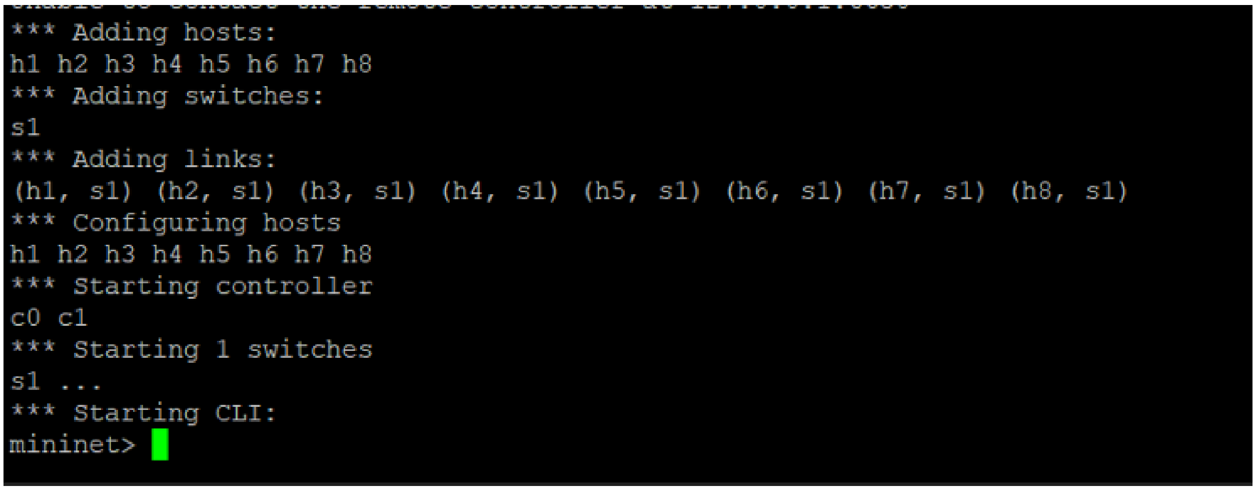


```
mininet@KidistMVirtualBox:~$ sudo mn --topo single,8 --mac --arp --controller=remote,port=6633 --controller=remote,port=6630 --switch=ovs,protocol=OpenFlow10
*** Creating network
*** Adding controller
```

Fig A- 1 Creating Multi controller Topology

- ➢ **sudo:** a command to run as a root user of all privileges
- ➢ **mn**: a command to set up a mininet emulator with sudo command
- ➢ **--topo single, 8:** a command to create a linear topology with 8 nodes in the mininet emulator.
- ➢ **--mac:** Auto set MAC addresses
- ➢ **--arp**: Populate static ARP entries of each host in each other
- ➢ **--controller,port** : software defined controller with remote options on predefined port.
- ➢ **--switch**=ovs,protocols=Open Flow10: Open flow switch with Open flow protocol 10.

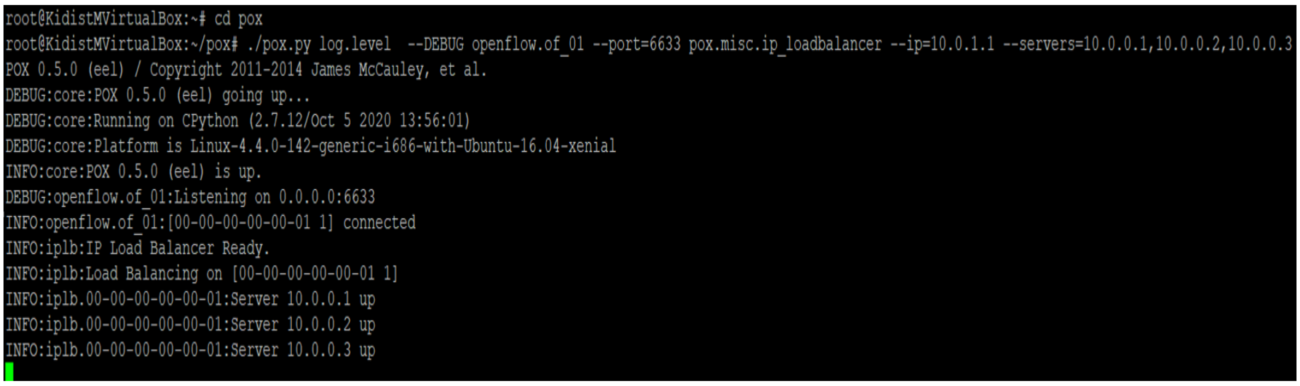
A.2. Connecting to the two remote controllers, adding links, starting controllers & switches in mininet



```
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1) (h7, s1) (h8, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0 c1
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Fig A- 2 Connecting to Multi-Controller and adding hosts

A.3. Running Random Load balancing algorithm on SDN POX controller one

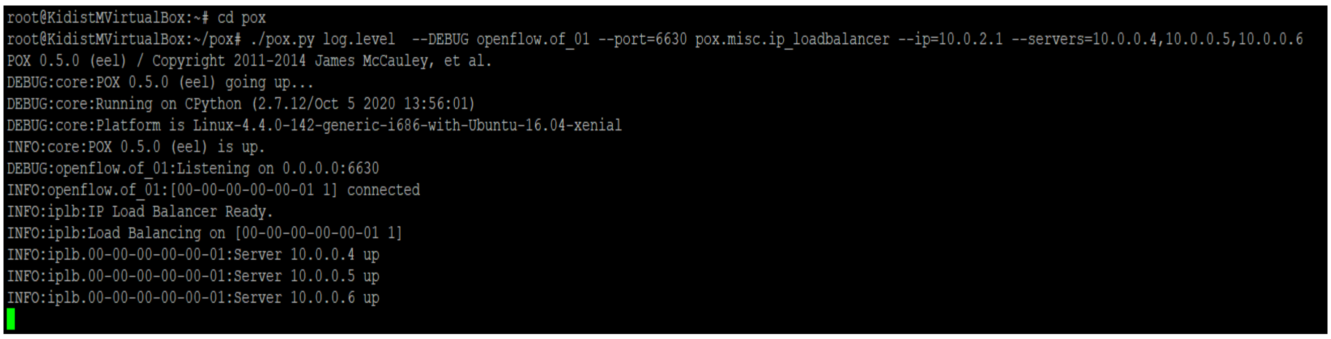


```
root@KidistMVirtualBox:~# cd pox
root@KidistMVirtualBox:~/pox# ./pox.py log.level  --DEBUG openflow.of_01 --port=6633 pox.misc.ip_loadbalancer --ip=10.0.1.1 --servers=10.0.0.1,10.0.0.2,10.0.0.3
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.12/Oct 5 2020 13:56:01)
DEBUG:core:Platform is Linux-4.4.0-142-generic-i686-with-Ubuntu-16.04-xenial
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:iplb:IP Load Balancer Ready.
INFO:iplb:Load Balancing on [00-00-00-00-00-01 1]
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.1 up
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.2 up
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.3 up
```

Fig A- 3 Running Random Load Balancing Algorithm on Controller one

- ➢ /POX.py : run POX controller
- ➢ Log. Level --DEBUG : log level as DEBUG open flow messages
- ➢ Open Flow.of_01 –port: connect to remote controller on port 6633 using Open Flow protocol
- ➢ Misc.ip_loadbalancer : is a random load balancing algorithm defined in POX controller
- ➢ --ip : create Virtual IP(VIP) for load balancer
- ➢ --servers: create servers with listed ip addresses.

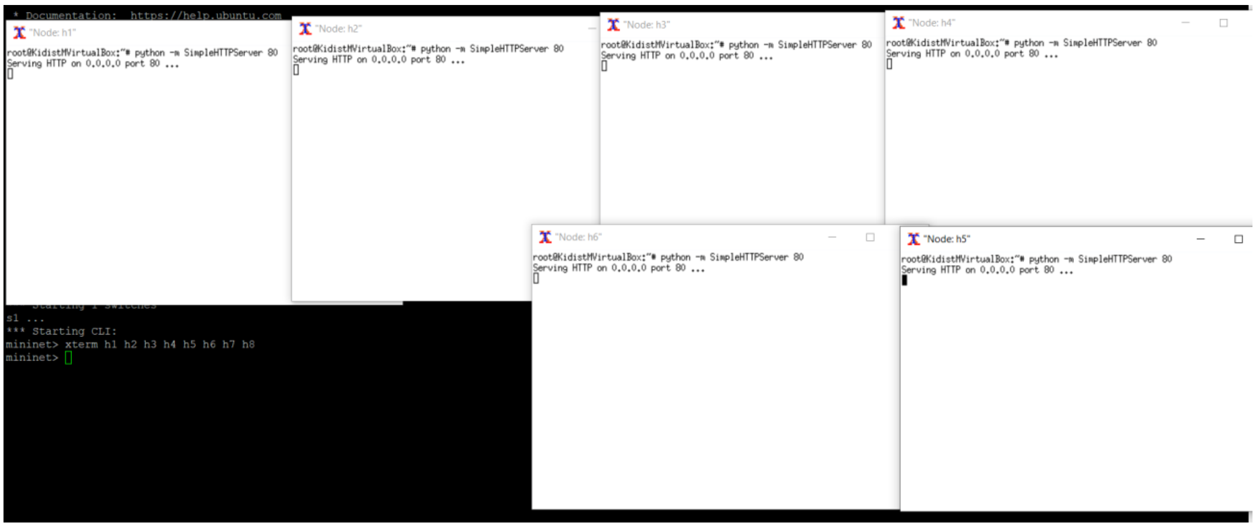
A.4. Running Random Load balancing algorithm on SDN POX controller Two



```
root@KidistMVirtualBox:~# cd pox
root@KidistMVirtualBox:~/pox# ./pox.py log.level  --DEBUG openflow.of_01 --port=6630 pox.misc.ip_loadbalancer --ip=10.0.2.1 --servers=10.0.0.4,10.0.0.5,10.0.0.6
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.12/Oct 5 2020 13:56:01)
DEBUG:core:Platform is Linux-4.4.0-142-generic-i686-with-Ubuntu-16.04-xenial
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6630
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:iplb:IP Load Balancer Ready.
INFO:iplb:Load Balancing on [00-00-00-00-00-01 1]
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.4 up
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.5 up
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.6 up
```

Fig A- 4 Running Rando Load balancing algorithm on controller two

A.5. Creating hosts as HTTP server



```
* Documentation:  https://help.ubuntu.com
s1 ...
*** Starting CLI:
mininet> xterm h1 h2 h3 h4 h5 h6 h7 h8
mininet>
```

```
root@KidistMVirtualBox:~# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

```
root@KidistMVirtualBox:~# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

```
root@KidistMVirtualBox:~# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

```
root@KidistMVirtualBox:~# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

```
root@KidistMVirtualBox:~# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

```
root@KidistMVirtualBox:~# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

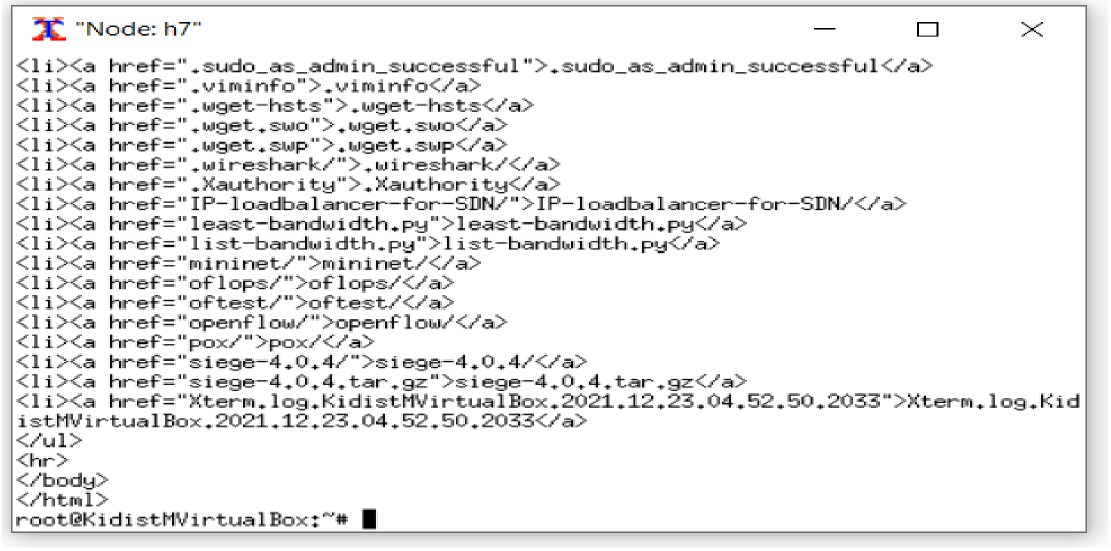
Fig A- 5 creating 8 hosts using Xterm

- ➢ Eight hosts are created using Xterm
- ➢ Six of the hosts are acting as an http server running on port 80 using python –m SimpleHTTPServer 80 & command.
- ➢ When two of the hosts are acting as a client generating concurrent HTTP requests.

A.6. Connecting client nodes with http server at port 80 using curl command

Among the eight hosts created h7 is connected to the first load balancer using this command

Curl 10.0.1.1

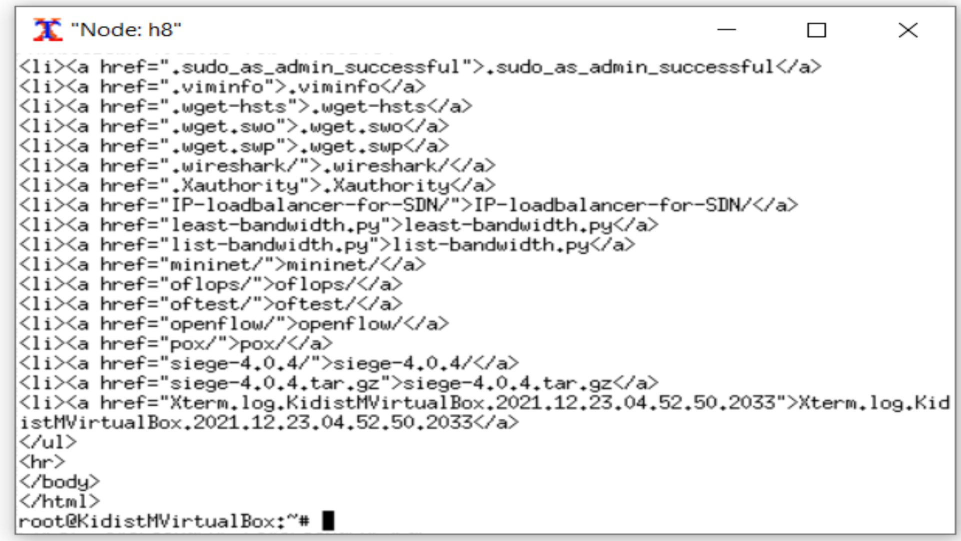


```
<li><a href=".sudo_as_admin_successful">.sudo_as_admin_successful</a>
<li><a href=".viminfo">.viminfo</a>
<li><a href=".wget-hsts">.wget-hsts</a>
<li><a href=".wget.swo">.wget.swo</a>
<li><a href=".wget.swp">.wget.swp</a>
<li><a href=".wireshark/">.wireshark/</a>
<li><a href=".Xauthority">.Xauthority</a>
<li><a href="IP-loadbalancer-for-SDN/">IP-loadbalancer-for-SDN/</a>
<li><a href="least-bandwidth.py">least-bandwidth.py</a>
<li><a href="list-bandwidth.py">list-bandwidth.py</a>
<li><a href="mininet/">mininet/</a>
<li><a href="oflops/">oflops/</a>
<li><a href="oftest/">oftest/</a>
<li><a href="openflow/">openflow/</a>
<li><a href="pox/">pox/</a>
<li><a href="siege-4.0.4/">siege-4.0.4/</a>
<li><a href="siege-4.0.4.tar.gz">siege-4.0.4.tar.gz</a>
<li><a href="Xterm.log.KidistMVirtualBox.2021.12.23.04.52.50.2033">Xterm.log.Kid
istMVirtualBox.2021.12.23.04.52.50.2033</a>
</ul>
<hr>
</body>
</html>
root@KidistMVirtualBox:~# █
```

Fig A- 6 connecting h7 as a client node

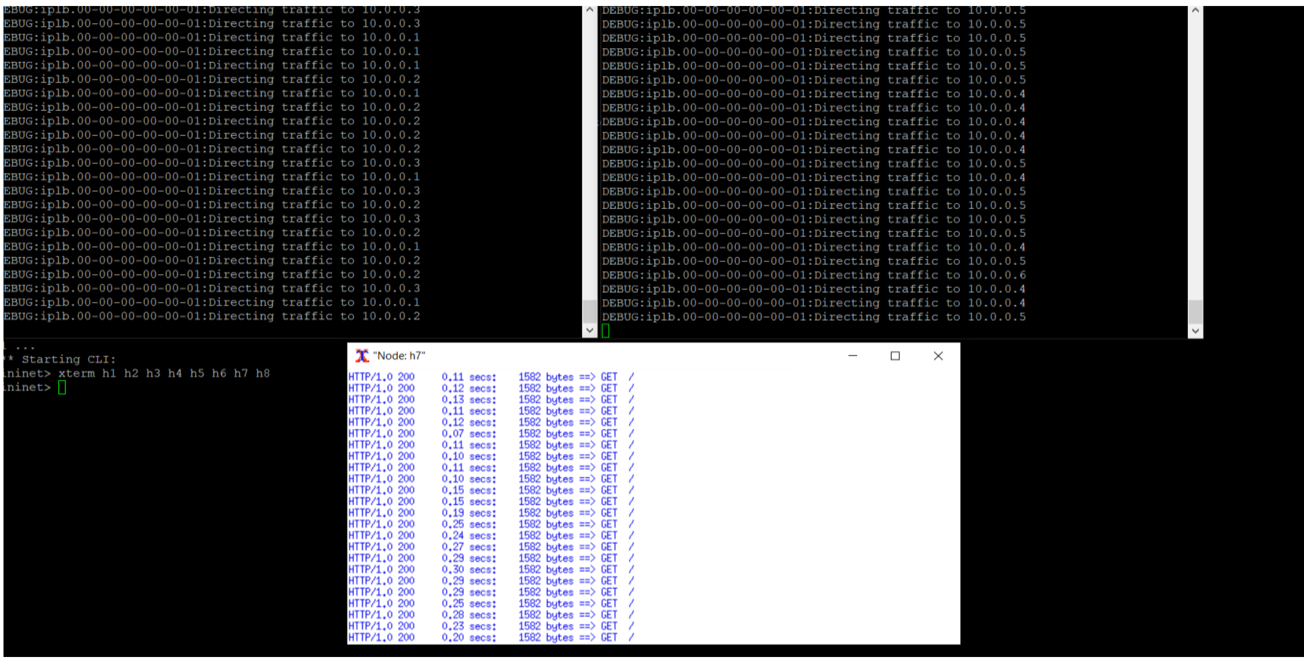
A.7. Connecting client nodes with http server at port 80 using curl command

Host eight (h8) is connected to the second load balancer using this command curl 10.0.2.1



```
<li><a href=".sudo_as_admin_successful">.sudo_as_admin_successful</a>
<li><a href=".viminfo">.viminfo</a>
<li><a href=".wget-hsts">.wget-hsts</a>
<li><a href=".wget.swo">.wget.swo</a>
<li><a href=".wget.swp">.wget.swp</a>
<li><a href=".wireshark/">.wireshark/</a>
<li><a href=".Xauthority">.Xauthority</a>
<li><a href="IP-loadbalancer-for-SDN/">IP-loadbalancer-for-SDN/</a>
<li><a href="least-bandwidth.py">least-bandwidth.py</a>
<li><a href="list-bandwidth.py">list-bandwidth.py</a>
<li><a href="mininet/">mininet/</a>
<li><a href="oflops/">oflops/</a>
<li><a href="oftest/">oftest/</a>
<li><a href="openflow/">openflow/</a>
<li><a href="pox/">pox/</a>
<li><a href="siege-4.0.4/">siege-4.0.4/</a>
<li><a href="siege-4.0.4.tar.gz">siege-4.0.4.tar.gz</a>
<li><a href="Xterm.log.KidistMVirtualBox.2021.12.23.04.52.50.2033">Xterm.log.Kid
istMVirtualBox.2021.12.23.04.52.50.2033</a>
</ul>
<hr>
</body>
</html>
root@KidistMVirtualBox:~# █
```

Fig A- 7 Sharing a load on Open flow based Multi-Controller



```
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
EBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
```

```
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.4
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.4
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.4
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.4
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.4
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.4
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.4
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.6
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.4
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.4
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.5
```

```
l ...
** Starting CLI:
ininet> xterm h1 h2 h3 h4 h5 h6 h7 h8
ininet>
```

```
HTTP/1.0 200     0.11 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.12 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.13 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.11 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.12 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.07 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.11 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.10 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.11 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.10 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.15 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.15 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.19 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.25 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.24 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.27 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.29 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.30 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.29 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.29 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.25 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.28 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.23 secs:    1582 bytes ==> GET  /
HTTP/1.0 200     0.20 secs:    1582 bytes ==> GET  /
```

Fig A- 8 running the two load balancers to share a client load on Multi-Controller