Evaluation of Load Balancing Technique on Web Servers Using Round Robin

Olanrewaju Lawrence ABRAHAM

LCU/PG/000905

Being a M.Sc. Thesis Submitted to the Department of Computer Science, Faculty of Natural and Applied Sciences, Lead City University, Ibadan, Oyo State, Nigeria

In Partial Fulfilment of the Requirements for the Award of Master Degree (M.Sc.) in Computer Science

2022

Certification

This is to certify that Olanrewaju Lawrence ABRAHAM with matriculation number LCU/PG/000905 carried out this research work titled "Evaluation of Load Balancing Technique on Web Servers Using Round Robin" in the Department of Computer Science, Faculty of Natural and Applied Sciences, Lead City University Ibadan, Oyo State for the award of Master of Science Degree in Computer Science and that this has been previously submitted.

_____ _____

Dr. A.A Waheed
(Supervisor)

Date

_____ _____

Dr. W. Sakpere
(Head of the Department)

Date

Dedication

This Project is dedicated to my Creator for His mercy and privilege upon my life, without which I can achieve nothing and also my Godly parent.

Acknowledgement

I will like to express my deepest gratitude to the Management and staff of Lead City University for granting me the opportunity to study further in a conducive learning environment.

I want to specially express my appreciation to my competent Supervisor, Dr. A.A Waheed for his guidance, motivation and relentless effort towards accomplishing this project. I want to thank you for being very patient and helpful to me all through the phases of my M.Sc. Programme. I equally want to express my appreciation to all the Lecturers in the Department of Computer Science for their immense contributions during my Course Work, Proposal defense, Pre-field and Post-field, thank you all for your encouragement and for sharing your resources and acquired knowledge with me.

I am indebted to Dr. (Mrs.) F.I Oyeyinka and my Boss, Dr. I.K Oyeyinka, Rector, Gateway (ICT) Polytechnic Saapade, Ogun State for their effort, financial support and useful advice to run this program successfully, thank you for been there for me Sir/Ma. To my Godly Parents, loving sisters and dearest friends who are always there for me, thanks for your care and prayers day and night.

"Even though the above-mentioned Institution and persons have assisted in the process of this research work, I alone stand responsible for the errors, if any, found in the work"

Abstract

Traditional networking architectures have many significant limitations that must be overcome to meet modern IT requirements. To overcome these limitations; Software Defined Networking (SDN) is taking place as the new networking approach. One of the major issues of traditional networks is that they use static switches that cause poor utilization of the network resources. Another issue is the packet loss and delay in case of switch breakdown. This project proposes an implementation of a dynamic load balancing algorithm for SDN based data center network to overcome these issues. In a data center environment, the load balancer is an integral part of the networking ecosystem. The primary function of a load balancer is to distribute traffic among a cluster of servers such that a single server does not become over-utilized and ensure that critical services keep running. Software Defined Networking (SDN) offers a cost-effective and flexible approach in implementing a load balancer. Moving away from traditional hardware-based networking approach, this project implements load balancing with the help of software. The SDN approach reduces the cost, offers flexibility in configuration, reduces time to deploy, provides automation and facilitates building a network without requiring the knowledge of any vendor-specific software/hardware. A test-bed has been implemented using Mininet software to emulate the network, and OpenDaylight platform (ODL) as SDN controller. Python programming language is used to define a fat-tree network topology and to write the load balancing algorithm program. Finally, iPerf is used to test network performance. The network was tested before and after running the load balancing algorithm. The testing focused on some of Quality of Service (QoS) parameters such as throughput, latency, bandwidth, delay, jitter, and packet loss between two servers in the fat-tree network. The algorithm increased throughput with at least 35.8%, and also increased the network utilization within the system.

Keywords: Application Programmable Interface (API), Load Balancer, OpenFlow, Round Robin, Software Defined Networking (SDN).

Word Count: 300

Table of Contents

List of Figures

List of Tables

List of Acronyms

Abbreviation	Meaning
ACL	Access Control List
API	Application program interface
CPU	Central Processing Unit
FSM	Finite State Machine
HPC	High-Performance Computing
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IS-IS	Intermediate System to Intermediate System
IT	Information Technology
LAN	Local Area Network
LLB	Link Load Balancing
MAC	Media Access Control
MPLS	Multi-Protocol Label Switching
NE	Network Elements
ODL	OpenDaylight
ONF	Open Networking Foundation

OPEX	Operational Expenditure
OS	Operating System
OSPF	Open Shortest Path First
QoE	Quality of Experience
QoS	Quality of Service
RR	Round Robin
WRR	Weighted Round Robin
SCTP	Stream Control Transmission Protocol
SDN	Software Defined Networking
TCL	Tool Command Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Virtual Machine

Chapter One

Introduction

1.1 Background to the Study

Due to the ongoing growth of web applications, there has been an increase in demand for online services and information. Given that all of these services are web-based and run on servers, we may presume that consumers can accurately and quickly find the majority of the information they need for their everyday lives online. Web servers therefore take in client requests, process them, and deliver the answers. Additionally, the effectiveness and efficiency of web servers impact their ability to attract businesses and web developers with features like precise and quick client responses. On the other hand, the demand placed on web servers has an immediate impact on their performance. Controlling the strain on the web servers is therefore essential. Therefore, it is crucial for researchers to design and propose an effective system that could handle a large amount of demand. The massive and ongoing increase in client requests for the services offered by the servers is another major contributor to the overload. Accordingly, server overcrowding impairs a company's ability to attract customers, lowers revenue, and damages its reputation[1]. Establishing a cluster server is yet another similar approach that is frequently utilized in businesses to improve performance. However, using a number of servers to increase resource availability and persistency necessitates a system to evenly balance and disperse the incoming demand among servers. This approach also aids in reducing the amount of traffic that is routed toward a single server inside the server cluster. In modern web technology, load balancing is a crucial component in setting up and offering a reliable service. In the meantime, load balancing

approaches boost the functionality of cluster servers by enhancing traffic management, reducing response times, increasing throughput, and minimizing the burdensome strain on cluster servers[2].

A load balancer, in general, functions similarly to a "traffic controller" for all server and routes traffics to an accessible one that can do so effectively. This guarantees that requests are met quickly and that no server is overworked to the point where performance is compromised. The load balancer helps an organization choose which server can effectively handle the requests in an effort to satisfy the application demands. Better user experience is produced as a result. The load balancer manages the flow of request between the servers and the clients by assisting servers in moving data effectively. Additionally, it evaluates the server's ability to handle requests, and if needed, the balancer eliminates the unfit servers or machine till their operations are fully restored. Traffic are sent to the active servers or machines when a failure is experienced on some servers and as such tagged as been offline, and when a new server is launched, requests are immediately forwarded to it[3]. To determine which load balancing strategies work best in each situation, a number of them will be tested and their impact on the web server analyzed.

The control plane and data plane are combined in a typical network architecture. As opposed to the SDN techniques, which divide and abstract the network management into control and data plane. The packets are transferred via the network through the data plane. The control plane is not implemented by the underlying switches, unlike conventional networks. With its intelligence, the control plane can issue network-wide commands across to the data planes. The control plane, which manages all of the routing choices made by the data plane, is a piece of software or a logical object. As a result, the network becomes nimble and directly programmable. Popular protocol adopted in SDN networks for connecting the controller with other network elements is called OpenFlow. Without mandating manufacturers to disclose the in-depth mechanisms of their

network devices, it happens to be an open standard that offers a standardized method to enable researchers to conduct experiments. Although they are distinct, OpenFlow and the SDN idea are frequently mixed together. OpenFlow is simply a protocol used to transmit information from the control layer across to the network elements, whereas SDN is the architecture that separates the layers[4]. There are numerous projects using OpenFlow, including controllers, virtualized switches, and testing software. Utilizing a network load balancer is important to increase the amount of available bandwidth, increase performance, and enhance redundancy. Network load balancing refers to the capability to balance network request directed from various internet connections. This feature stabilities all network sessions like Web, email, etc. and increases the overall bandwidth throughput available by distributing each client bandwidth consumption across a number of connections.

1.2 Statement of the Problem

Web server application performance, web traffic, and congestion management become issues due to the rise in web users as demands on internet services and information grow steadily as a result of continual expansion of web applications. For a network to transmit data with high throughput and minimal delay, network resources must be managed dynamically. Static switches are used in conventional networks. These networks have the drawback that every flow/request follows a single, preset route through the network[5]. Packets typically drop when a switch fails until a replacement routing is chosen. Poor network resource use, where alternate links to the destination are inactive, is another problem.

1.3 Aim and Objectives of the Study

This study is aimed at enhancing the performance of web server using load balancing techniques in SDN-based data center networks in which a higher level of performance can be achieved.

The aim is to be achieved by the following objectives: to

i. enumerate various load-balancing methods for handling network request distribution.
ii. adopt the most appropriate network load-balancing technique.
iii. evaluate the effect of the load balancing technique on the network.

1.4 Significance of the Study

The primary contribution of this study is it to improve the effectiveness, performance, and dependability of web servers and so contributes to the maximization of customer satisfaction.

1.5 Scope of the Project

In order to deliver and maintain high throughput and low latency on the network, this study is meant to distinguish, experiment, and assess the effect of load balancing algorithms utilizing key performance indicators based on multiple parameters.

1.6 Operational Definition of Terms

Internet: Internet is a global communication in term of electronic network that links computer networks and structural computer facilities.

Network Congestion: When we have too many communications moving through the internet at the same time, this is known as congestion network. Or Congestion of Network happens when the amount of packets being transmitted across the network exceeds the network's packet handling capability[6].

Internet Protocol: The mechanism or procedure by which data is transmitted from a computer to the other over the internet is known as Internet Protocol. Each computer on the Internet (known as a host) has one IP address that distinguishes it from all other computers on the network.

Algorithms: An algorithm is a procedure or collection of rules that a computer uses to do calculations or other problem-solving actions[7]. A set of instructions for solving a problem or completing a task is known as an algorithm.

Data Communication: Data communication refers to the interchange of information between a sender and a receiver across a transmission means such as a wire connection.

Throughput: Throughput can be referred to as the total slice of packets successfully acquired through the sink node per unit time[8]. To design an efficient algorithm, the throughput obtained should be high.

1.7 Outline of the Study

The remaining parts of the document are laid down in the following way: The previous relevant research on SDN and network load balancing is presented in **Chapter Two**. The components utilized in this study to set up the testbed are presented in **Chapter Three**, along with detailed report of the load balancing algorithm utilized. The scenarios that were proposed

are discussed in **Chapter Four**, and the results of the implementation are shown. Conclusions and future potential research areas are presented in **Chapter Five**

Endnotes

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

[2]. S. Khan, F.K. Hussain & O.K. Hussain, "***Guaranteeing End-to-End QoS Provisioning in SOA Based SDN Architecture: A Survey and Open Issues," Future Generation Computer Systems,*** **2021**, vol. 119, pp. 176–187.

[3]. H. Mokhtar, X. Di, Y. Zhou, A. Hassan, Z. May & S. Musa, "***Multiple Level Threshold Load Balancing in Distributed SDN Controllers," Computer Networks,*** **2021**, vol. 198, pp. 108369.

[4]. M. Hasan, N. A. Zakaria & Z. Z Abidin, "***Load Balancing Algorithms in Software Defined Network," International Journal of Technology and Engineering,*** **2019**, vol. 7, no. 6S5, pp. 686-693.

[5]. B. Alankar, G. Sharma, H. Kaur, R. Valverde & V. Chang, ***"Experimental Setup for Investigating the Efficient Load Balancing Algorithms on Virtual Cloud," Sensors,*** **2020**, vol. 20, no. 7342, pp. 1-26.

[6]. M. Alotaibi & A. Nayak, ***"Linking Handover Delay to Load Balancing in SDN-based Heterogeneous Networks," Computer Communications,*** **2021**, vol. 173, pp. 170-182.

[7]. I. Alam, K. Sharif, F. Li, Z. Latif, M.M. Karim, S. Biswas, B. Nour & Y. Wang, ***"A Survey of Network Virtualization Techniques for Internet of Things Using SDN and NFV," ACM Computing Surveys,*** **2020**, vol. 53, no. 2, Article 35, pp. 1-40.

[8]. T. Guesmi, A. Kalghoum, B.M. Alshammari, H. Alsaif & A. Alzamil, ***"Leveraging Software-Defined Networking Approach for Future Information-Centric Networking Enhancement," Symmetry,*** **2021**, vol. 13, no. 441, pp. 1-19.

Chapter Two

Literature Review

2.1 Conceptual Review

2.1.1 Concept of Load Balancing

In order to evenly disperse traffic over a network, specific hardware was used to launch the load balancing idea in 1990. Load balancing improved in security with the advent of Application Delivery Controllers (ADCs), allowing programs to be accessed without interruption even during periods of high demand. ADCs can be divided into three types: native software load balancers, virtual appliances, and hardware appliances. The software-based ADCs are utilized in this age of cloud computing to carry out functions just like their hardware counterparts, but with higher scalability, capability, and flexibility. Generally a typical load balancer functions as a server's "traffic controller" allotted the duty of routing requests to an available server that can successfully fulfill them[1]. This guarantees that requests are met quickly and that no server is overworked to the point where performance is compromised. The Load Balancer helps to further determine which server is active and can effectively handle the incoming requests as part of an organization's effort to satisfy the application demands. Better user experience is produced as a result. The load balancer manages information distribution between the server and the respective endpoint devices by assisting servers in moving data effectively[2]. Additionally, the server's ability to handle requests is evaluated, also when required, the load balancer excludes unstable or faulty server by making them inactive until it is fixed. Requests are sent to the other servers

when one server goes offline, and when a new server is launched, requests are immediately forwarded to it.

2.1.2 Definition of a Server

A server refers to a computer system also known as the host that makes resources, information, services, or programs available across a network to other computers, sometimes known as clients. Theoretically, computers are regarded as servers whenever they share resources with client devices.

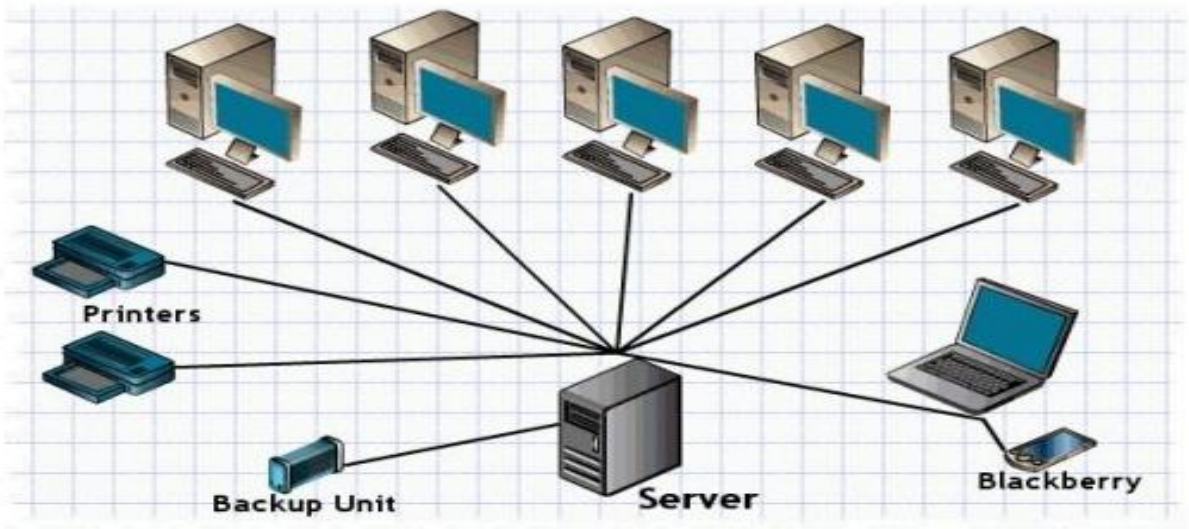

Figure 2.1: Server connection

(Source: https://www.itrelease.com/2021/04/what-is-client-server-network-with-example/)

2.1.3 Types of Server?

File Server: A file server is a device that stores and shares files. Shared files stored on a server may be used by numerous clients or users. Furthermore, storing files centrally makes backup and fault tolerance solutions easier to implement than trying to guarantee the safety (from either hardware or software hazards) and integrity of documents on every device in a company. For increased performance, hardware for file servers might be developed to enhance transfer process.

Print Server: The management and distribution of printing capabilities is made possible by print servers. A print server is capable of handling printing demands from all clients instead of adding a printer to each workstation. There is no longer a need for a second computer-based print server because modernized large and even expensive printers have in-built print server. Internal print server works by responding to requests from individual clients.

Application Servers: Resource consuming apps utilized by many people are frequently run by application servers. This eliminates the requirement that every client have enough resources to run the programs. Additionally, it thus eliminates the need for software maintenance on multiple client systems as opposed to just one.

Domain Name System (DNS) Servers: are application servers that convert human-readable names into computer understandable internet addresses[3]. The domain name system is a broadly dispersed repository of names and additional domain name servers, each of which can be utilized to demand an additional computer name that is not otherwise known. When a client requires a system's address, a DNS request is forwarded to a DNS server along with the name of the required resource. The required IP address is returned by the DNS server from its table of names.

Mail Server: One popular class of application server is the mail server. Emails sent to a user are received by mail servers, which keep them on file until a client requests them on the user's behalf.

The availability of an email server makes network connectivity possible and easier for a device to be set up.

Web Server: A web server is one of the most prevalent server kinds on the market right now. An intranet or the Internet can be used to access applications and data that are hosted on a web server, a particular type of application server[4]. Web servers respond to requests for web contents or other web-oriented services made by client computers' installed web browsers. Web servers like Apache, Microsoft Internet Information Services (IIS), and Nginx are frequently used.

Database Server: Businesses, users, and other services use a surprising quantity of data. Databases are used to store a lot of that data. Multiple clients must be able to access databases at once, and they can use a staggering amount of disk space. Database servers manage database programs and respond to many client requests. Oracle, Microsoft SQL Server, DB2, and Informix are examples of common database server programs.

Virtual Servers: Virtual servers have completely taken over the server industry. Virtual servers function only within specialized software called a hypervisor, which make it different from traditional servers which are designed to be installed as an OS on machine hardware. Each hypervisor may run thousands of virtual servers at once. As though it were actual physical hardware, the hypervisor delivers virtual hardware to the server. The virtual server utilizes only the virtual resources allotted to it, however the hypervisor transfers the necessary computing and storage requirements to the base hardware, which is shared by all other virtual servers[5].

2.1.4 Web Server Application

A web server contains both software and hardware resources that responds to client requests forwarded via the World Wide Web through the HTTP (Hypertext Transfer Protocol) and other

protocols. The primary function of a web server is to produce website content to users by storing, processing, and transmitting web pages to them. SMTP (Simple Mail Transfer Protocol) and FTP (File Transfer Protocol), which are used for email services, file transfer, and storage, are also supported by web servers in addition to HTTP[6]. While web server software manages accessibility to hosted files, web server hardware connects to the internet and enables data interchange with other connected devices. The client/server model is exemplified by the web server operation. Web server software is a must for all server engines that houses websites. Web hosting, also known for hosting data meant for web addresses and web-based applications, or web applications, all of which are stored on the web server. Websites' domain names are used to access web server software, which makes sure that the content of the site is sent to the user who requests it. There are various parts to the software side, including at least one HTTP server. Both HTTP and URLs can be understood by the HTTP server. A web server is a piece of hardware that houses web server software and other website-related assets like HTML texts, pictures, and JavaScript files. A web browser, such as Google Chrome or Firefox, will use HTTP to request a file that is stored on a web server. The HTTP server will accept the request after the web server receives it, find the requested content, and provide it to the browser through HTTP.

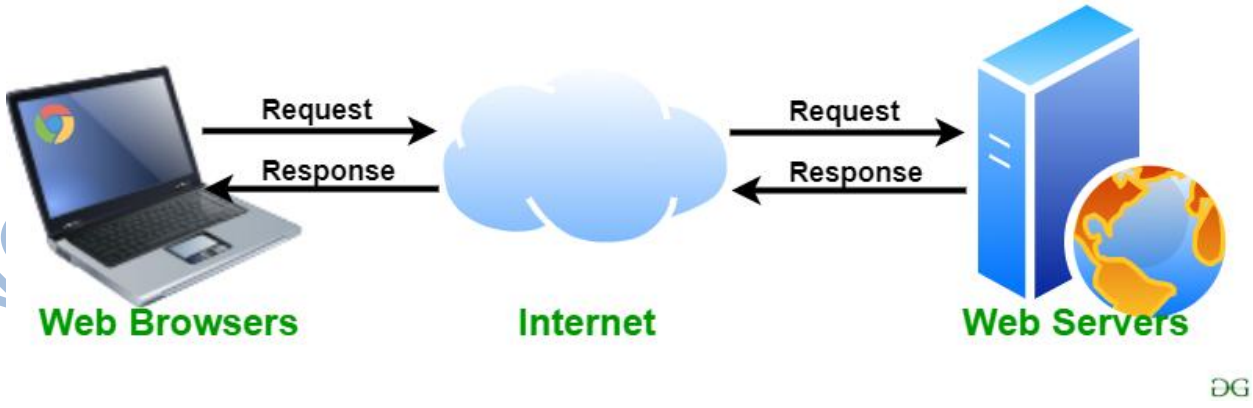

Figure 2.2: Client-Server connection

(Source: https://www.geeksforgeeks.org/web-server-and-its-type/)

More specifically, a browser will go through a number of processes while requesting a document from a web server. An individual will first enter a URL in the address bar of a web browser. The web browser will then translate the URL through the DNS (Domain Name System) or by looking in its cache to find the IP address of the domain name. The browser will access a web server as a result. The appropriate file will then be requested by the browser via an HTTP request to the web server. In response, requested contents will be returned through HTTP once more. Error messages are returned when the requested page does not exist or when an error was encountered during the process. Response from the web servers are seen on the browser. Multiple domains can be contained on one web server.

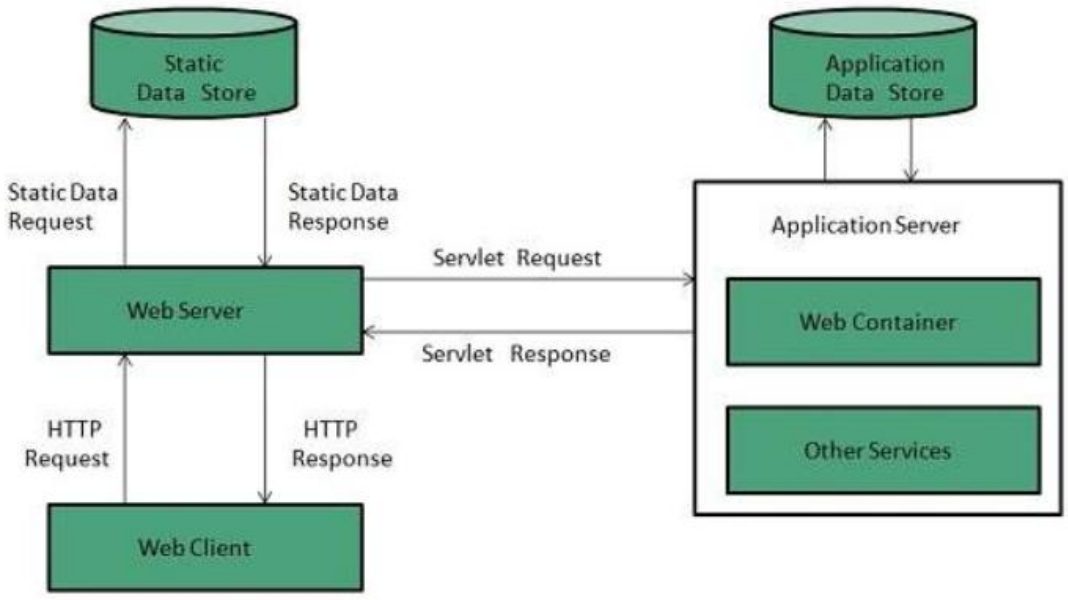

Figure 2.3: Request and Response Handling

(Source: https://www.tutorialspoint.com/internet_technologies/web_servers.htm)

2.1.5 Web Server Software

- Apache HTTP Server: It is a free and open source web server designed to function on all major operating system like Windows, Mac OS X, Unix, Linux, Solaris, and other operating systems that was created by the Apache Software Foundation; it requires the Apache license.
- Microsoft Internet Information Services (IIS): It was created by Microsoft for Microsoft platforms; while being extensively used, it is not open sourced.
- Nginx: a popular open source web server among administrators due to its scalability and little resource consumption[7]. Its event-driven architecture enables it to support numerous concurrent sessions. In addition, Nginx may function as a load balancer and proxy server.
- Lighttpd: an open-source web server included with the FreeBSD operating system. It uses minimal CPU resources and is regarded as speedy and secure.
- Sun Java System Web Server: a Sun Microsystems free web server that may be used with Windows, Linux, and UNIX. It is capable of handling medium-sized to huge websites.

2.1.6 Web Application Architecture

Web application architecture defines the interconnections amidst applications, databases, and middleware systems all located on the web server. It guarantees that several apps operate at once. Let's use the example of opening a web page to illustrate this concept. When a user types a URL into the address bar of a web browser and clicks Go, that specific web address is requested. In response to the request, the server transmits files to the browser. After that, the browser runs the files to display the requested page. The user can finally engage with the website. The code that the web browser parses is the most crucial point to keep in mind here. A web application

operates similarly. There may or may not be precise instructions in this code that tell the browser how to react to the various forms of user input[8]. As a result, a web application architecture must have all of the supporting components and external application interfaces for the full software program, in the example above, a website.

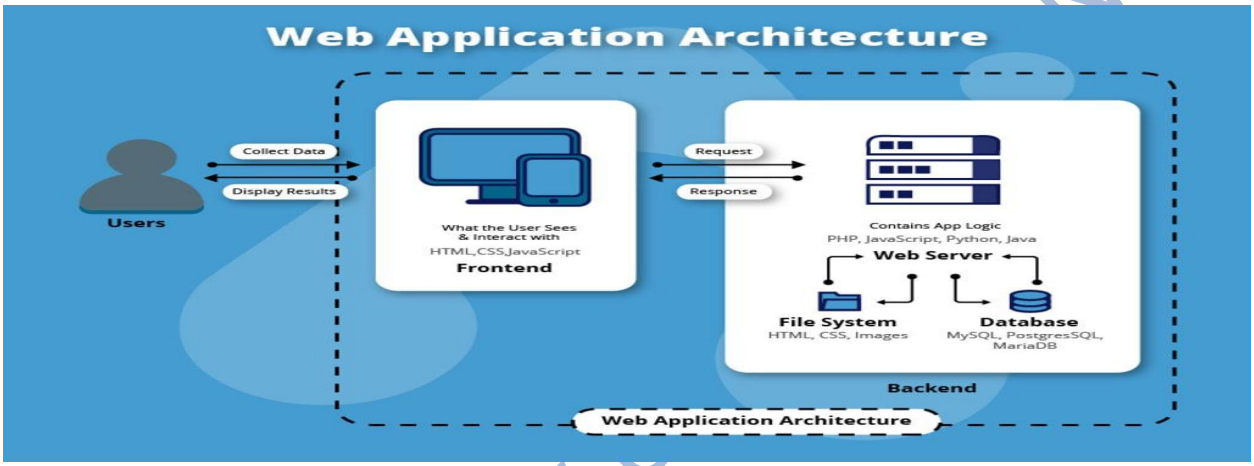

Figure 2.4: Web Application Architecture

(Source: https://vocal.media/01/web-application-architecture)

In the present world, web-based communication is used by the majority of apps and devices, a significant amount of global network traffic, and the web application architecture itself[9]. A web application architecture must address reliability, scalability, security, and robustness in addition to efficiency. Any typical web application has two separate programs (codes) running simultaneously. Which remain:

• Client-side code – These are codes or scripts that runs within browsers and reacts to user input

• Server-side code (SSC) - SSC are server-side codes the handles and implement the business logic and in responds to the HTTP requests.

Actions of the server is predefined by the web developers responsible for creating the web application. Server-side languages not limited to C#, Java, JavaScript, Python, PHP, Ruby, etc. are utilized during server-side programming. A server can be used to run any program that can reply to HTTP requests. The user's requested page is created and various forms of data, such as user profiles and user input, are stored by the server-side code. The end-user never sees it. The client-side environment is created using a markup and styling languages like CSS, HTML, and JavaScript. The web browser parses this code. Client-side code, as opposed to server-side code, is viewable and modifiable by the user[10]. It responds to input from users.

The client-side code cannot directly access server files; it can only connect with the server through HTTP requests.

2.1.7 Web Application Components

Components of a web application can refer to any of the following:

• **UI/UX Components** of a web application include dashboards, notifications, settings, analytics, and activity logs, among other things. These components have no impact on the operability of the web application architecture. Rather, they remain part of the interface layout strategy for a web program.

• **Structural Components** – comprises of the client and server.

• **Client Component** – Frontend languages are the development tools used in creating the client component. No modification is required to the user's operating system or device. The client component signifies the functionality of a web application interacted with by the end users.

• **Server Component** – backend or programming languages and frameworks are used to in creating the server component[11]. Application logic and database are represented in the server components. The application logic handles all the functions and commands used, meanwhile the database contains/store all of the data used.

2.1.8 Models of Web Application Components

A web application's model is chosen based on one of the following three possible options:

1. One Web Server, One Database

It is the simplest straightforward and least trustworthy model. A single server and database are both used in this model. Such a model guarantees that a web app will crash alongside an occurrence of a server crashes. Therefore, it isn't very trustworthy. Real web applications don't often employ this approach. It is mostly used to carry out test projects and to learn and comprehend the foundations of the web application.

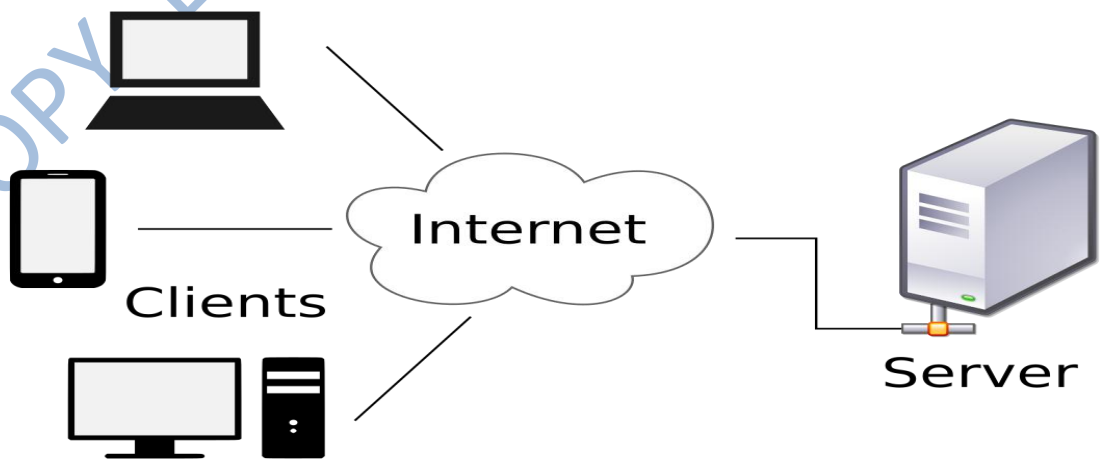

Figure 2.5: One Server Model

(Source: https://www.google.com)

2. Multiple Web Servers, One Database

In this paradigm, no data is kept on the webserver. The database, which is administered externally to the server, is updated when the webserver receives data from a user, processes it, and as well stores such data on the database. The term "stateless architecture" is also sometimes used to describe this. This web application component type requires a minimum of two web servers. All of this is done to prevent failure. In occurrence of a web server crash, the other one will take over. The web app will continue to run while all requests are automatically forwarded to the new server. In light of this, dependability is improved over the single server with a single database paradigm. However, in the event that the database becomes corrupted, the web application will also become corrupted.

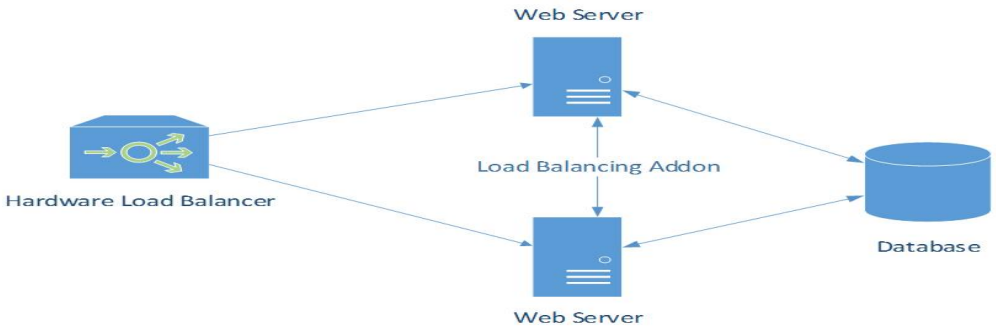

Figure 2.6: Multiple Web- Server, One Database Model

(Source: https://www.progress.com/documentation/sitefinity-cms/reference-architecture-diagrams)

3. Multiple Web Server, Multiple Databases

In a scenario where the databases or the webservers do not support a single point of failure, it is the most effective component paradigm for online applications. For this kind of model, there are two alternatives. To either evenly distribute the data among the used databases or to store the same data in each database. In the former scenario, there is often no need for more than two databases, whereas in the latter, there is a possibility that some data could become unavailable in the event that the database were to crash[12]. However, both situations make advantage of DBMS normalization. It is recommended to deploy load balancers when the scale is high, such as when there are more than 5 web servers, databases, or both.

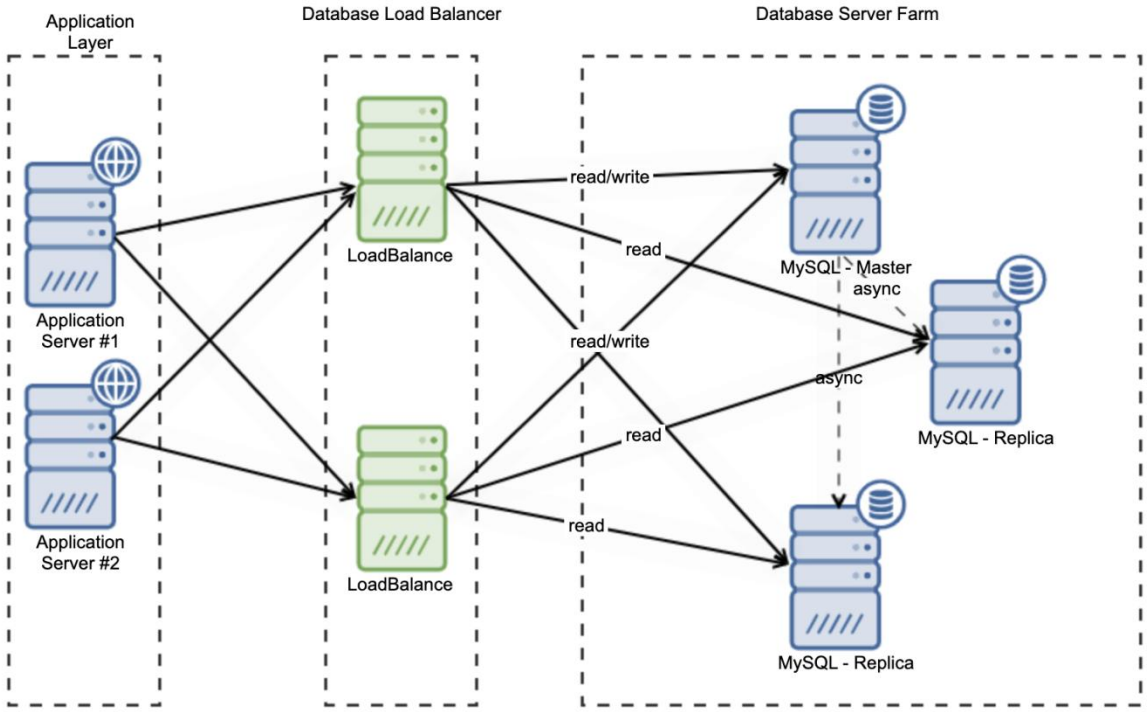

Figure 2.7: Multiple Web- Server, Multiple Database Model

(Source: https://severalnines.com/database-blog/how-does-database-load-balancer-work)

2.1.9 Types of Web Application Architecture

An interaction pattern between different web application components is known as a web application architecture. The distribution of the application logic between the client and server sides determines the architecture adopted.

The three main forms of web application architecture are as follows. The following is an explanation of each of them:

- **SPAs (Single-Page Applications):** enable dynamic user interaction by supplying updated content on the current page rather than downloading entirely new pages from the server each time. The basis for aiding dynamic pages and also making SPAs a reality is AJAX, a condensed form of Asynchronous JavaScript and XML. Single-page applications resemble traditional desktop programs in that they don't allow disruptions to the user experience. SPAs are created in such a way that they ask for the majority of required content and information elements. This results in the acquisition of an interactive and intuitive user experience.
- **Microservices**: are streamlined, compact services that carry out a particular function. With the help of the microservices Architecture framework, developers may increase productivity and accelerate the deployment process altogether[13]. An application built utilizing the microservices architecture has components that are not reliant on one another. As a result, developers using the microservices architecture chooses independently their preferred technology stack. It facilitates quicker and easier application development.
- **Serverless Architectures** - the server and other frameworks are granted to a third-party cloud facility services provider. This method has the advantage of letting applications run code logic without having to worry about infrastructure-related activities. When the web application development business are not ready to undergo the stress of infrastructure management, the serverless architecture is the ideal option.

2.2 Performance Measurement Parameters

The effectiveness and availability ratio of a system or process are, in fact, indicated by measurement parameters. The health of the web server can be efficiently ensured in the interim by using measures for observation and assumption activities, combining and continually controlling the hardware features of web servers (such as reliability, availability, scalability resource usage etc.)[14]. The measures that the majority of authors take into account when assessing the current trends are given. These parameters are crucial for creating and implementing a load balancing algorithm. These measurements are used to evaluate an algorithm's performance on web servers[15]. The taxonomy diagram below summarizes the metrics used in the available research and divides them into two groups: qualitative and quantitative.

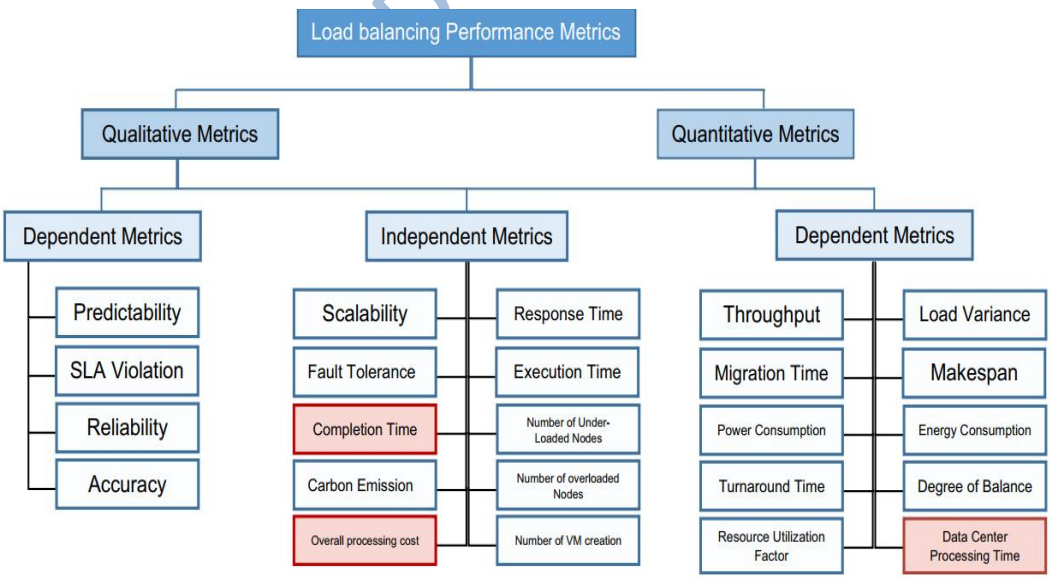

Figure 2.8: Load Balancing Performance Metrics

(Source: https://google.com)

- **Resource Utilization (RU):** measures how fully a system is using its resources, such as memory and CPU. Its purpose is to gauge the RU level within the server farm. RU is necessary on the occasion of a rise in service demand. A maximum RU is necessary for the load balancing algorithm to function well.
- **Scalability (S):** An algorithm should operate successfully in the face of any unforeseen conditions, much like a system does. In other words, the algorithm should continue to scale no matter how many tasks or how much work is put on it. For the load-balancing algorithm to work well, it needs to be highly scalable.
- **Throughput (TP):** is the total number of request successfully processed in a certain amount of server time. It represents the volume of information been transported between two locations. For the load balancing algorithm to work well, it needs to have a high TP. It speaks of the requests that were fulfilled or handled at a specific moment in time[16]. Usually, it is used to measure how busy a server is. In other words, it measures the server's bandwidth adaptation to determine the data transmission rate.
- **Response Time (RT):** is known to be the time utilized by the algorithm to complete a task. Timing which includes waiting, transmission, and servicing times are all considered. It refers to how long it takes to answer a user question. A decent load balancing algorithm must have a minimum RT. Response time, often known as latency, is the result of both the waiting period and the response period.
- **Makespan (MS):** The time taken to complete a task and as well provide feedback. It is a crucial metric in the cloud environment's scheduling process. It is used to calculate how long it

takes to complete a collection of tasks. A decent load balancing algorithm must have a minimum MS.

- **Fault Tolerance (FT):** A load balancing system should be able to function properly even if some system components fail. Hence, if one server is overcrowded, tasks should still be able to be completed on another server that is available. High FT for the load balancing algorithm's performance.
- **Migration Time:** The duration required to move a job from one server to another is known as the migration time (MT). The system's availability shouldn't be impacted during the migration process. The cloud's virtualization concept is incredibly important. Low MT for the load balancing algorithm to perform well.
- **SLA Violation:** Indicates the quantity of SLA violation elements that have been reduced in terms of deadline restriction, priority, etc. When resources (VMs) are unavailable because they are overloaded, SLA violations occur. A lower minimum SLA would increase user happiness

2.3 The SDN Concept

The concept of software-defined networking (SDN), targeted at bringing more innovations into the network management process as compared to the widely known but rigid current approach, has lately attracted the attention of researchers in the aspect of programmable networks and as well realign the focus of the networking community into same[17]. Due to the great amount of complexity involved, designing and managing computer networks may become an extremely difficult undertaking. A network's management and evolution are complicated by the close coupling between the control plane, where decisions about traffic management are

decided, and the data plane, where real traffic forwarding actually occurs. Network administrators must manually translate highly ranked decisions into low level instructions, making such very difficult and error prone in complicated networks. Introducing new protocols have been known to be a very slow procedure that requires many years of standardization, phased deployment and testing to ensure a seamless integration with other needed implementations provided by different vendors, bringing a form of dynamicity into the network, such as intrusion-detection systems and load balancers, all of this requires altering the infrastructure and logical layout of the network[18].

By encouraging innovation within the network administration as well as distribution of network amenities through the programmability of the core network entities making use of some type of generic network API, the concept of network programming was released as a measure to tackle some of this issue. Comparing the analogue models to the use of programmable networks tells of how the new system have out-performed the existing system as the new system allows users perform a range of functions without modifying the hardware resources, this wish have proven to have produces the desired results according to the needs of the user[19]. As shown in Figure 2.9, SDN is a quite recent concept of a programmable network which give a new view of network planning and its operation by offering an abstraction which separates the control from the data plane. Represented within the method, the hardware devices such as routers, switches, etc. are only responsible for packets forwarding to their destinations in accordance with the controller's instructions, which are usually a set of packet-handling rules[20]. The controller saddled with the responsibilities of making necessary decisions in which it has a general overview.

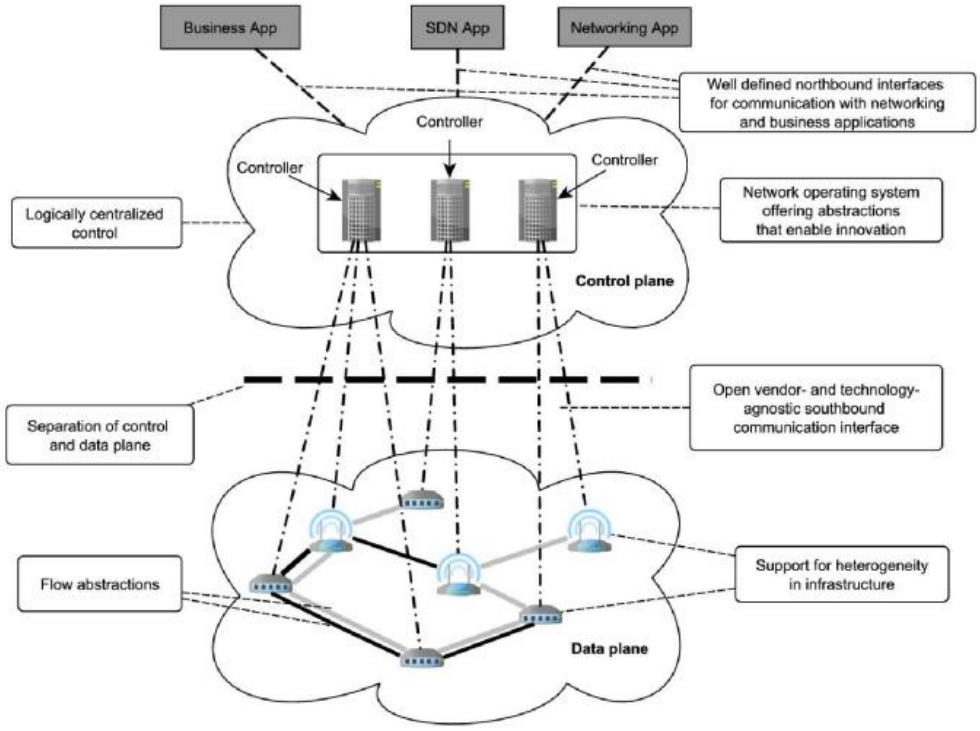

Figure 2.9: SDN Controller

(Source: https://google.com)

It has been considered to have immeasurably simplify in diverse ways the network evolution and management, this which was achieved by the approach of logically separating the centralized control plane away from the data plane, this which has quickly been the focus of intense research attention in the networking field. As a result of separating both planes, applications and protocols can be tested and distributed through the network without harming unrelated network traffic, the process which has been observed to have incorporated diversity into it also allows addition of new infrastructure devoid of difficulty, middle boxes implementation via the software control, as well as allowing new potential fix to be presented for complications that have long been in the spotlight, like managing the extremely complex core of cellular networks[21]. Although the SDN

concept is relatively new, many of its fundamental concepts have existed for many years and have merely matured. The user will thereby have a better knowledge of the driving forces behind and potential alternatives to the solutions put out over time, which influenced the present SDN approach, by analyzing the history of programmable networks.

2.3.1 SDN History and Evolution

The word "programmable" refers to the idea of simplification of network organization as well as its restructuring, it's crucial to realize that in actuality it encompasses a variety of concepts that have been put out through time, each with a distinct focus and a unique method of accomplishing its objectives. In this context, the fundamental concepts that gave rise to SDN will be examined together with additional alternatives that were put up and had an impact on SDN's development but did not enjoy the same level of broad success.

2.3.2 Early History of Programmable Networks

The idea of programmable networks was first introduced in the mid-1990s, as was already said, just as the Internet was beginning to gain significant popularity. Up until that point, only a few features which includes electronic mail as well as file transfers, were used in conjunction with computer networks. The rapid expansion of the internet outside of research institutions caused the creation of massive networks, which sparked the concern of academics and developers in implementing and testing novel concepts for network services. The enormous complexity of operating the network infrastructure, however, soon became clear as a significant impediment in this direction[22]. Without even ensuring vendor interoperability, network enabled devices were deployed as "black boxes" built to sustenance a handful of protocols essential to the network's functionality. Due to this, it became impossible to change the control logic of such

devices, which significantly constrained network evolution. Numerous initiatives were made to address this issue, with the goal of developing more open, extensible, and programmable networks. The Open Signaling (OpenSig) working group and the Active Networking effort came up with two of the most significant early concepts that suggested methods of isolating the control software from the underlying hardware and offering open interfaces for administration and control[23].

OpenSig - The Open Signaling working group was established in 1995 with the goal of implementing the programmability principle within the ATM networks. The primary concept was the division between networks' control and data planes, with signaling amidst the two planes carried out via an open interface. Due to this, it's now possible to remotely manage and configure ATM switches, effectively transforming the network in its entirety into a networked platform and substantially streamlining the method of introducing new services[24].

The concepts for open signaling interfaces promoted by the OpenSig community served as inspiration for additional study. In order to facilitate this, the Tempest framework, which operates on the OpenSig concept, allows several switch controllers to simultaneously manage multiple switch partitions and, as a result, to operate multiple control architectures over the same physical ATM network. Due to the fact that network operators were not needed to create a single unified control architecture meeting the control requirements of all future network services, they were given more latitude[25]. DCAN was a different effort that sought to build the framework required for controlling ATM networks (Devolved Control of ATM networks). The basic concept was to remove the ATM network switches' control and administration capabilities from the hardware and allocate them to separate, external workstations. DCAN assumed that multiservice networks' control and management functions were naturally distributed because it

was necessary to distribute resources along a network route so as to assure QoS. In order to provide any further management capabilities, such as stream synchronization in the management domain, the connection between the management entity and the network was carried out using a minimalistic protocol, similar to what present SDN protocols like OpenFlow do. Midway through 1998, the DCAN project was formally completed.

Active Networking - first emerged mid 1990s and received significant funding from DARPA. Similar to OpenSig, its primary objective remained the development of programmable networks to support network developments. The primary concept underlying active networking is that resources on network nodes are made available over a network API, enabling network specialist active control of the nodes through running arbitrary programs. As a result, active networking, opposing the fixed functionality provided by OpenSig networks, allows for the quick disposition of adaptive services and the dynamic setup of networks during execution. On active nodes, the typical active network architecture specifies a three-layer stack. The NodeOS project and Bowman are two well-known examples of the several projects that provide various NodeOS implementations. One or more deployment environments that serve as template for creating active networking applications, such as ANTS and PLAN, exist with the next layer[26]. The active applications, or code created by network administrator, are the final component of the top layer.

The active networking community works with two programming models: the capsule model, where the executable code remains embedded in standard data packets, and the programmable router/switch model, where the executable code to be run at the network nodes is formed using out-of-band procedures. The capsule approach emerged as the most inventive and closely related to active networking of the two. The rationale for this is that it provided a

straightforward technique of deploying new data plane capability across network channels, offering a fundamentally different approach to network management[27]. Since many of the principles used in SDN were developed by the active networking community, both prototypes had a substantial effect and left a lasting legacy.

2.3.3 Evolution of Programmable Networks to SDN

2.3.3.1 Shortcomings and Contributions of Previous Approaches

Although the main ideas of these pioneering techniques projected programmable systems with the intent of supporting advancement and foster open networking settings, all of which never found widespread acceptance. The absence of convincing difficulties that these systems were able to tackle was one of the key causes of this failure. Although the idea of network programmability appeared to improve the performance of some instances, such as information circulation and network administration, there was no genuine compelling requirement that would make the switch to the new paradigm necessary and prompt the commercialization of these early ideas. Active networking and open signaling's incorrect user group focus is another factor in why they failed to get widespread adoption. Up until that point, only programmers employed by the manufacturers who created the network devices could modify them. Although end user programmers were actually a very uncommon use case, the proposed prototype argued that amidst its benefits was the ease in usability it would offer users to program the network. This obviously had a bad effect on how programmable networks were perceived by the research institutes and, more crucially, the industry, since it obscured their benefits for those who could genuinely profit from them, such as internet service providers and network operators. Additionally, a lot of the early techniques were more concerned with fostering data-plane

programmability than that of the control-plane. The disintegration of the control from the data plane was one of the fundamental concepts underlying programmable networks, but most solutions that were put out did not make this distinction explicitly[28]. The control plane, which perhaps offers more potential than the data plane for the discovery of convincing use cases, was hampered in any attempts at innovation by these two factors. Another factor that contributed to the early programmable networks' failure was their emphasis on developing novel designs, programming models, and suitable environment without consideration towards relevant applied concerns which includes performance as well as security provided. Even though these characteristics are not essential to network programmability, but play crucial role when it involves making this concept commercially viable. It is obvious that the aforementioned flaws of the early attempts at programmable networks were the barriers to their general success. However, these efforts were incredibly important since they defined fundamental ideas that changed how networks are thought of and highlighted promising new research fields. Even their flaws were of great importance since they exposed several weaknesses that needed to be fixed if the new paradigm was ever going to succeed.

2.3.3.2 Shift to the SDN Paradigm

Major developments in networking occurred throughout the first decade of the 2000s. High-speed Internet connectivity for customers is now possible thanks to new technologies like ADSL. At that time, it was more affordable than ever for the typical consumer to have access to an Internet connection, which could be utilized for a variety of purposes, including massive file transfers, teleconference services, multimedia, and e-mail. Networks saw cataclysmic

repercussions as a result of the widespread use of internet and other new services that came with it. Networks' size and scope grew as a result of the rise in traffic volumes. Industry players like ISPs and network operators began putting more emphasis on the performance of the network, stability, and quality of service, and needed improved methods for carrying out crucial network setup and administration tasks like routing, considered to be the best rudimentary in the current times[29].

The solutions to each of these issues represented strong use cases for programmable networks, which once again brought networking professionals and the industry's focus to this issue. The effort of shifting the control functions outside network devices was made easier as a result of advancement of servers, which turn out to be noticeably superior to the mechanism utilized in routers. This technological change led to the development of new, better attempts at network programmability, with SDN serving as the most notable example. The primary reason behind the accomplishment of SDN remains the ability to improve on the positive aspects of the first efforts at programmable networks while also resolving their flaws. The transition from early years of programmable networks up till the invention of SDN did not happen overnight; rather, as we will show in the next paragraphs, there were several intermediate steps[30]. The inability to definitely bring separation amidst the control and data planes of networking devices was, as was already indicated, one of the primary shortcomings of early attempts at programmable networking. By separating both planes, the Forwarding and Control Element Separation working group aimed to fix this through changing the internal design of network devices. In ForCES, two logical objects could be noticed: Control Element (CE), in charge of the logic of network devices, such as management protocol implementation and control protocol processing, and the Forwarding Element (FE), this which is targeted towards the data plane and in charge of per-packet

processing and conduct. The forwarding activities to the forwarding element as instructed by the control element was enforced by a standard interconnection protocol between the two parts.

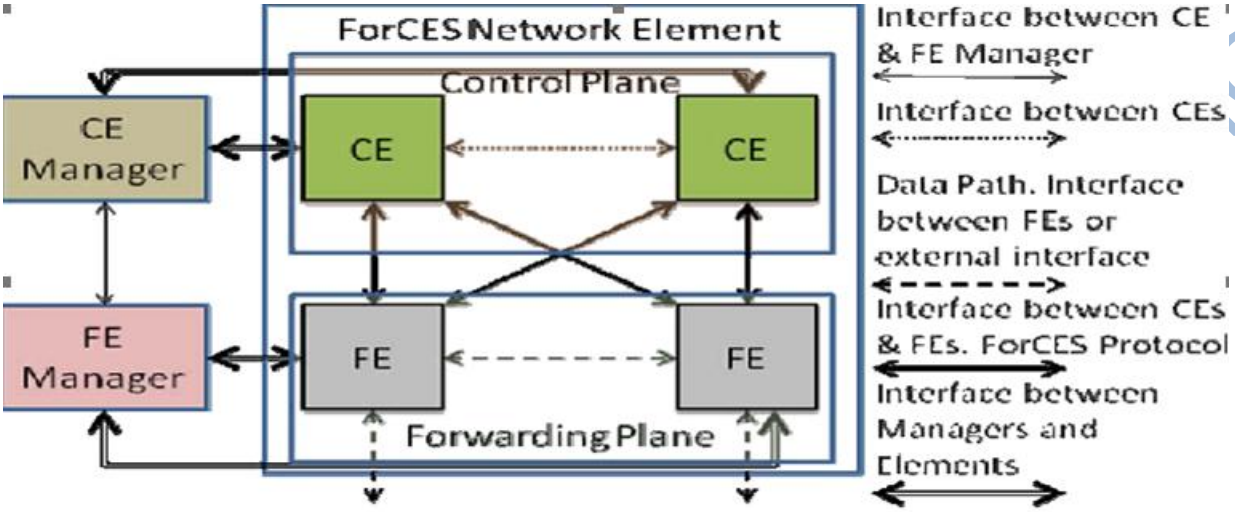

Figure 2.10: SDN ForCES framework

(Source: https://google.com)

ForCES was designed with the idea of letting the forwarding and control planes develop independently and by offering a standardized method of interconnection, it would be probable to create various FEs (general-purpose or specialized) combinable with other third-party controls, giving innovators more freedom to experiment. The 4D project was another strategy that aimed to clearly separates both control and forwarding components of network devices. Reasons for isolating the decision controller from the basic network components was underlined by 4D, as it was by ForCES. The 4D project, in contrast to other methods, proposed a design based on four different planes which includes: a decision plane in charge of configuring the network; a broadcasting plane in control of providing the decision plane with information on the view of the network; a detection plane that grants network devices ability to discover other close devices; and a data plane in charge of dispatching traffic.

2.3.3.3 The Emergence of Software Defined Networking

Scalable network experimentation became a topic of interest for funding organizations and researchers in the later part of the year 2000s. This importance was primarily sparked as a result of the necessity to introduce new protocols and services that aimed to improve QoS and performance in a very large network community. It was as well fueled by the achievement of experimental infrastructures like Planetlab and the creation of numerous programs. Large-scale testing was formerly a difficult process to complete; researchers were mainly restricted to employing imitation settings for assessment, which, while their usefulness, did not continuously include all the crucial network-related considerations in the similar way as an actual testbed would. The necessity for programmability, intended to make network administering and service setting out simpler and in same vein allow numerous testing to be run concurrently at the same infrastructure while each utilize different forwarding procedures, was a crucial prerequisite of such infrastructure-based efforts[31]. The Clean Slate Program was developed by a team of Stanford academics who were inspired by this concept. OpenFlow was suggested for academics trial protocols in common networking environments, which had as its goal to "reinvent the Internet." Like earlier strategies including ForCES, OpenFlow adhered to the idea of separating the control and forwarding planes and standardized information transfers amongst the two by means of a straightforward communication protocol.

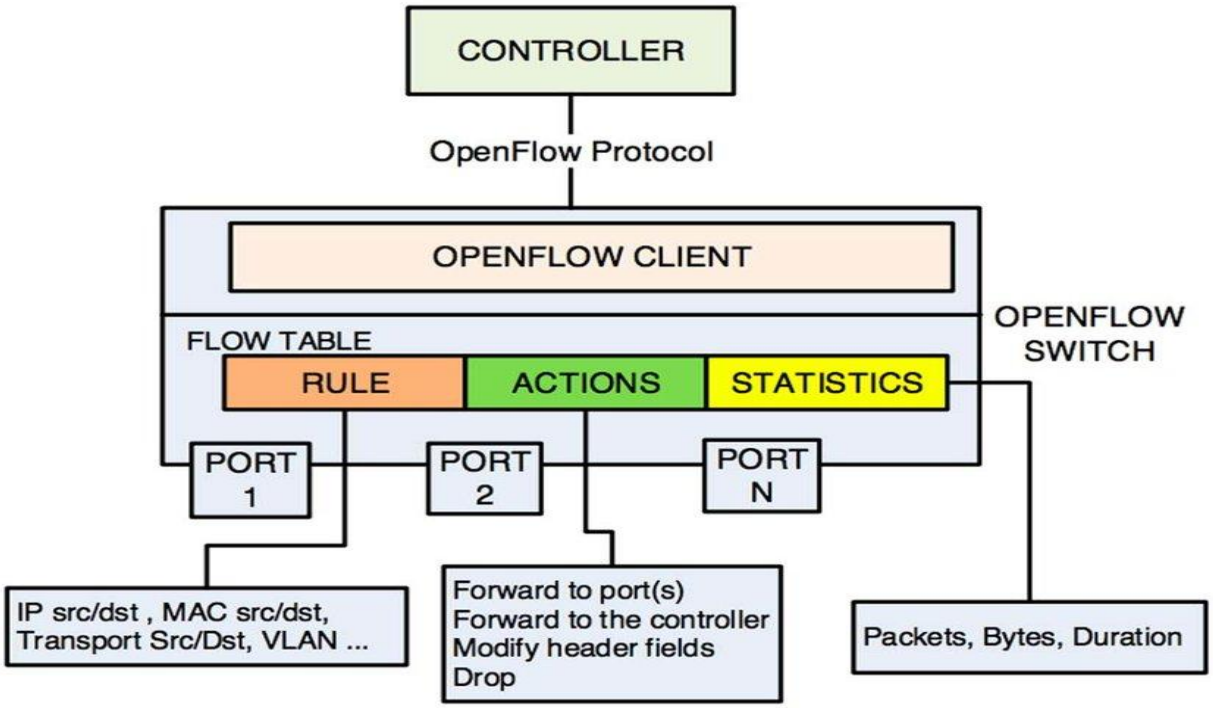

Figure 2.11: SDN OpenFlow Protocol

(Source: https://www.researchgate.net/figure/Communications-via-OpenFlow-protocol)

OpenFlow's solution, which offered architectural compliance for network programmability, inspired the term "SDN" to be coined to refer to all networks that adhere to the same architectural principles. When likened to the outmoded networking paradigm, the core notion behind SDNs remains building of horizontally linked systems by separating the control plane and the data plane despite offering an ever-more-complex set of concepts. We may get the conclusion that the road to SDN was indeed long, with many concepts being suggested, tried, and reviewed, propelling exploration in this area even further, by considering the indicators and significant programmable network experiments given[32]. SDN was not really a novel concept; rather, it was the optimistic outcome of the information and experience that had been distilled from many of the concepts discussed in this section. In contrast to these approaches, SDN was

able to incorporate the furthermost crucial network programmability models into a design that developed at the correct period and ensured many convincing use cases for interested parties. Although it is uncertain if SDN will represent the subsequent significant architype transferal in networking, the potential the aforementioned offers is unquestionably great.

2.3.4 SDN Paradigm and Applications

The SDN model, the most modern example in the advancement of programmable networks, is the subject of this section, which focuses on its fundamental principles. We must look at SDN from both a macro- and micro-level in order to comprehend the concepts behind it and the advantages that it promises to bring. For this, we first offer a broad overview of its design in this section before delving deeply into an analysis of its constituent parts.

2.3.4.1 Overview of SDN Building Blocks

The SDN approach, as was previously cited, enables network service management through the abstraction of lower level operations. Network managers do not need to deal with the low level specifics about how packets and flows are controlled by network devices; instead, they may leverage the abstractions provided by the SDN architecture. This is accomplished through the separation of the control from the data planes using a layered design. We can see the network devices such as wireless access points, routers, and switches in the data plane, which is the lowest layer. In the framework of SDN, these devices have remained devoid of all control logic (for example, routing algorithms like BGP), and instead implement a number of forwarding procedures for modifying network data packets and flows, offering an open, abstract interface for interfacing with the upper layers[33]. These components are frequently denoted to as switches in the SDN terminology. The next layer displays the control plane, which is where the controller is

located. This object is in charge of providing a programmatic environment to the system, needed to add new features and carry out different administration activities. It also encapsulates the networking logic. SDN's control plane, in contrast to earlier approaches like ForCES, is completely separated from the network gadgets and is thought to be logically integrated, though physically it can be integrated in one server or dispersed into multiple servers, which regulate the network structure as a whole. The concept of the network operating system perception has been established by SDN, which is a key feature that sets it apart from other attempts at programmable networks. Remember that earlier initiatives like active networking suggested using a node operating system (like NodeOS) to manage the underlying hardware. An easier interface for managing the network is revealed by a network OS, which provides a more general perception of the network state in the physical devices. In this abstraction, the applications interprets the network as a lone system under a logically centralized control architecture. In other words, control logic uses the system embedded operating system like an intermediary platform to maintain a steady observation of the system state, which is then broken down to deliver different networking routines like topology detection, routing, managing flexibility and statistics, etc[34]. The application stand point, which comprises of all requests that use the operations offered by the controller to carry out network-like responsibilities like load assessment, network virtualization, etc., is at the top of the SDN stack. One of SDN's key benefits is the plainness it offers to outside designers over the constructs it establishes for quick creation and innovation of new solutions in a variety of distributed contexts. Additionally, because their functionality can now be implemented as software applications that observes and alter the network status via the network OS facilities, dedicated middle wares like firewalls, intrusion detection solution are not necessary in the network topology thanks to the SDN architecture. Since it opens up an extensive

variety of potential for invention, the presence of this level clearly includes significant importance to SDN and makes it an appealing solution for both researchers and business. Finally, well defined interfaces allow the controller to communicate with the data and application layer planes (APIs). In the SDN architecture, there are two primary APIs that can be distinguished as:

i) Southbound Interface: controls the interaction amid the controller and the network setup; and

ii) Northbound Interface: defines a link amid the network boundary and the controller.

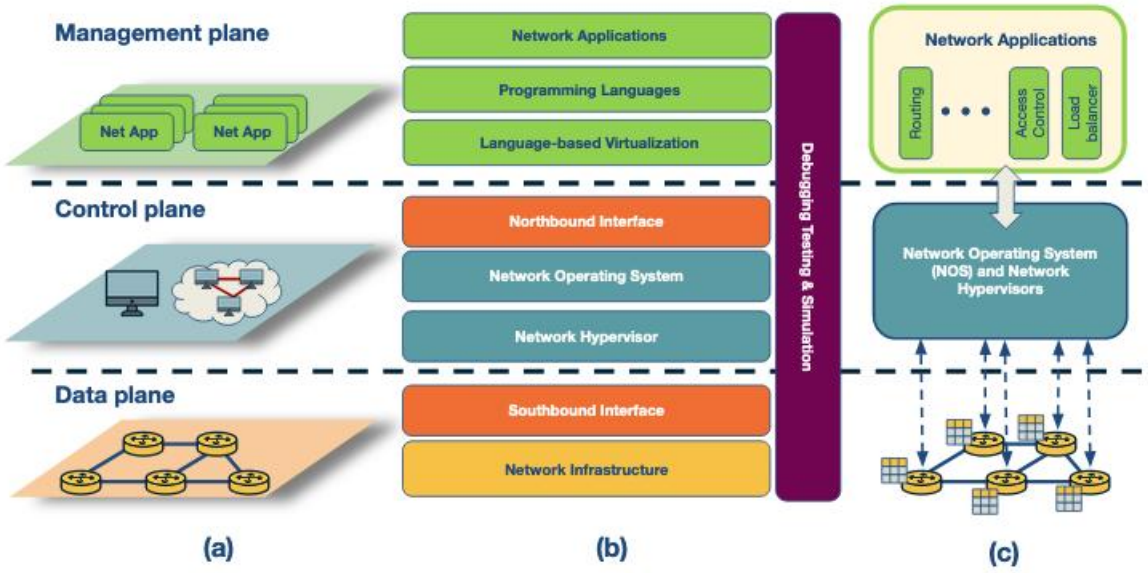

Figure 2.12: SDN Architecture

(Source: https://www.researchgate.net/figure/ SDN-architecture)

The connection between the hardware, operating system, and the user interface in most systems is accomplished in a manner similar to this. After going over the SDN architecture in a more

basic sense, it is now time to delve further into a study of each of the recently provided building components individually. The following section will examine a few SDN application examples.

2.3.4.2 SDN Switches

The network structure is regarded as the utmost essential component of the system in the orthodox networking model. Each network device contains all the functionality necessary for the network to function. A router, for instance, must have the necessary hardware, such as a Ternary Content Addressable Memory for swift packet dispatching as well as refined solution for running distributed channeling procedures like BGP. Similar to wired access points, wireless access points require the appropriate components for wireless communication as well as solutions for packet dispatching, imposing access regulation and forwarding rules[35]. Nonetheless, because network devices are closed, changing their behavior dynamically is not an easy operation. By separating control from forwarding activities, the three-layered SDN architecture described previously alters this and makes it easier to operate network devices. The hardware for keeping the dispatching tables, such as Application-specific integrated circuits is retained by all forwarding devices, as was already mentioned, but their logic is removed.

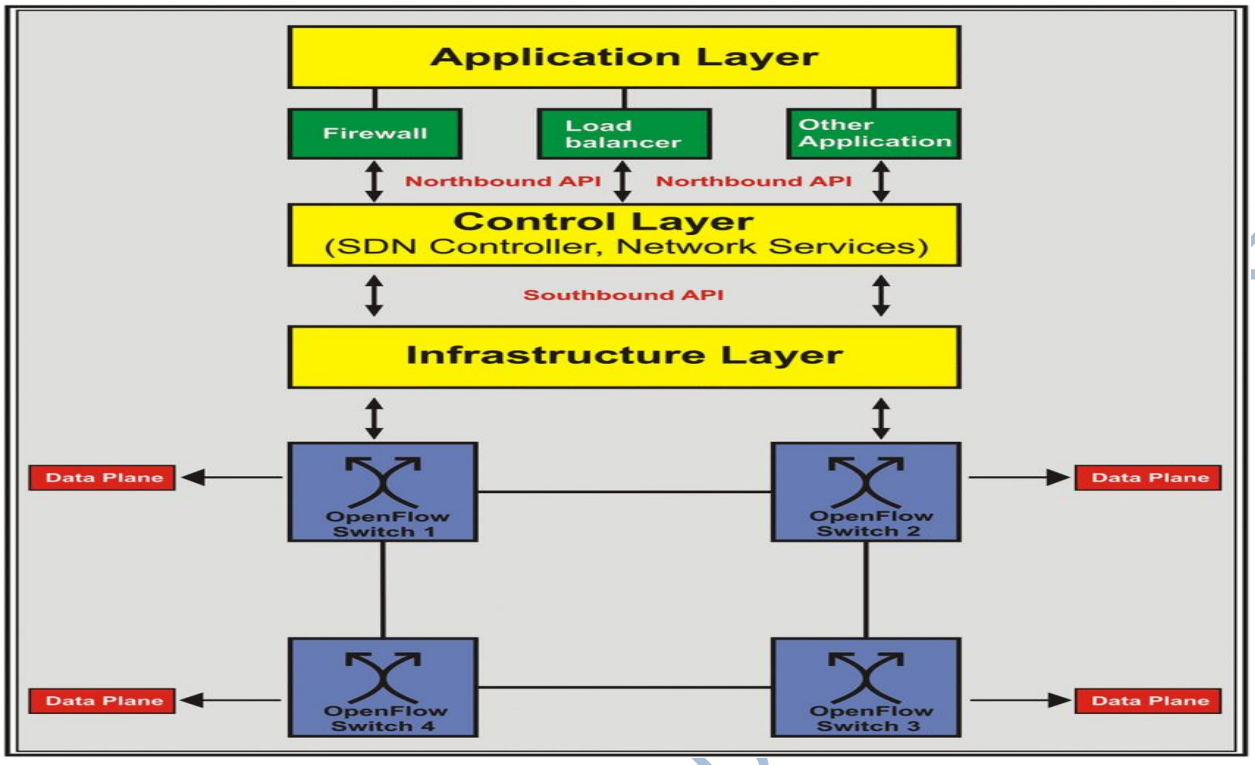

Figure 2.13: SDN Three layer distribution

(Source: https://www.researchgate.net/figure/A-three-layer-distributed-SDN)

By implementing new forwarding procedures over an abstract interface, the controller directs the switches on how to forward packets. A switch's forwarding table is reviewed each time a packet reaches the switch, and the packet is afterwards dispatched appropriately. Although a clear three-layered architecture for SDN was described in the aforementioned overview, it is still not clear where the lines should be drawn amidst the control or the data plane. For instance, even in the case of SDN switches, some procedures and scheduling setup are still seen as being a part of the data plane[36]. However, there is no intrinsic issue that prevents these tasks from being integrated into the control interface by adding some form of concept that enables the regulation of low level activities within the switching components. An advantageous strategy would be one that made it easier to implement new, more effective low level switch operation techniques. On

the other hand, while centralizing all control activities theoretically has the benefit of making network administration simpler, doing so may cause scalability problems if the controller's physical implementation is similarly centralized. Keeping part of the logic in the switches may therefore be advantageous. DevoFlow, an OpenFlow variant, divides packet distribution into two classifications: tiny ("mice") distribution controlled through the switches directly and large ("elephant") flows demanding the controller's involvement. Similar to the DIFANE controller, which is limited to the straightforward duty of splitting the procedures over the switches, intermediary switches are utilized to store the essential rules in the DIFANE controller[37]. The use of wildcards for packet forwarding and taking into account several attributes of the packet, such as the source and destination addresses, ports, application, etc., make SDN forwarding rules more compounded than those used in orthodox networks, which is another problem with SDN switches. As a result, managing packets and flows is difficult for the switching hardware. ASICs utilizing TCAM are necessary for the forwarding procedure to be quick. Unfortunately, because such specialized hardware is pricey and energy-intensive, each switch can only accommodate a certain amount of forwarding accesses for flow-based dispatching patterns, which limits the scalability of the network. This could be addressed by adding a supporting CPU alongside the switch or a proximate location to carry out both planes functionalities, by introducing new designs that are more expressive and enable the execution of more packet processing operations. The problem of hardware restrictions extends beyond fixed networks and affects the wireless and mobile sectors as well. As was done with the data plane of fixed networks, the wireless data plane necessities been rewritten to provide further beneficial constructs. The notion of splitting the control from the data plane is supported by the data plane constructs provided by OpenFlow[38]. It should be made clear that backwards compatibility is a crucial component for the acceptability

of the new paradigm, regardless of how SDN switches are implemented. Although there are pure SDN switches that have no incorporated control, the hybrid approach supporting both SDN and conventional process and procedures would likely be the most effective in these early stages of SDN. The infrastructure in most enterprise networks continues to use the conventional method, despite the fact that the properties of SDN showcase a convincing way out for various real-world applications. Consequently, a transitional hybrid network form would presumably make SDN easier.

2.3.4.3 SDN Controllers

One of the basic tenets of the SDN concept is that an operating system for networks should exist which differentiate the network structure from the application level. This network OS plays the role of organizing and handling the network's requirements and providing programs running on top of it with an abstract, unified picture of all its parts. This concept is similar to those used within a normal computer system, which has the operating system sits amidst the hardware and the user interface and also handles all hardware resources and giving user programs access to shared services[39]. Similar to how a conventional computer program developer would operate in this environment, network administrators and developers now have it at their disposal. In comparison to the traditional networking paradigm, the SDN model is more suitable to an extensive variety of applications and diverse network practices due to its logically consolidated control and comprehensive network construct. Considering a heterogeneous setting made up of a stationed network and a wireless network that are both made up of numerous connected network.

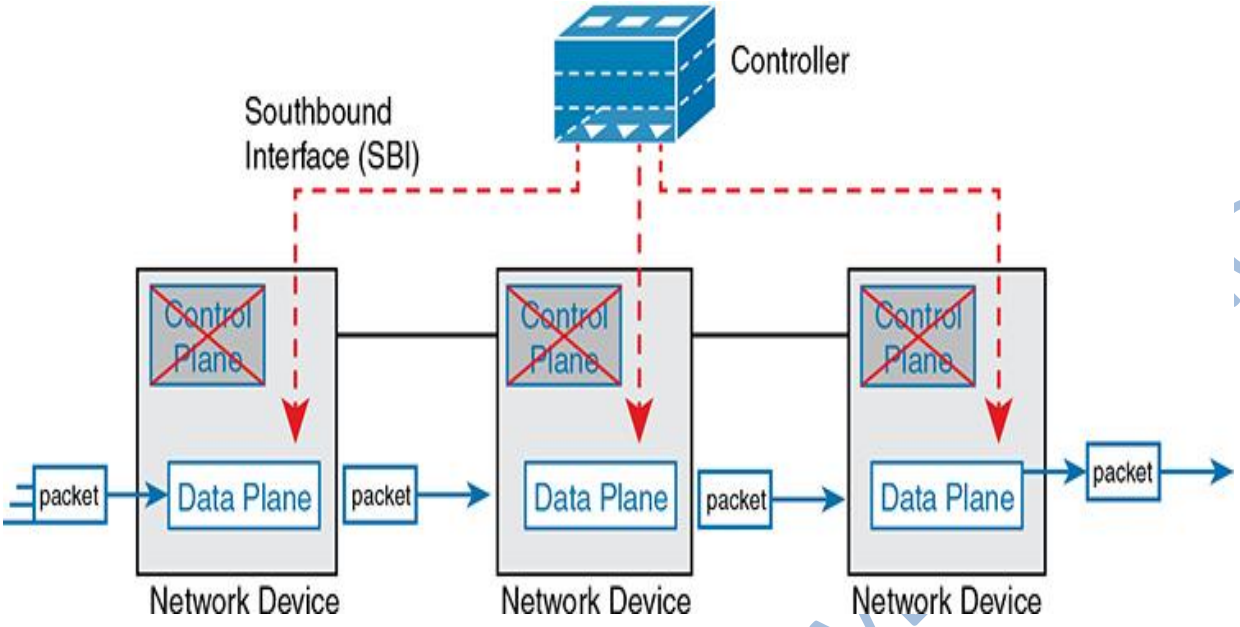

Figure 2.14: SDN Controller

(Source: https://www.ciscopress.com/articles/article.asp?p=2995354&seqNum=2)

Respective network gadget needs a unique low level setup by the network operator in the conventional networking paradigm in order to function effectively. Additionally, because respective gadget targets a dissimilar networking procedure, each one would consume a unique managerial and setup needs, necessitating additional work on the part of the admin to ensure that the network in its entirety functions as expected. Conversely, the administrator wouldn't need to worry about minute details with SDN's logically centralized control. The network OS would be responsible for interacting and setting up of the functionalities of the network gadgets while managing the network would be carried out by outlining a suitable highly priorities procedure[40].

2.3.4.4 Control Centralization in SDN

The SDN design stipulates that a central organization in charge of administration and policy enforcement logically controls the network infrastructure, as was already mentioned.

Numerous suggestions for controllers that are physically integrated, such as NOX and Maestro, have been made. Implementing a controller is made easier by a physically centralized control design. Since each switch is managed by a single physical entity, there are no consistency problems on the network, and every application sees the same state within the network. In spite of its benefits, this strategy has very similar flaw as all integrated systems, this infers the controller serves as the entire network's single point of failure[41]. The approach of integrating several controls to a switch can help you get around this problem by allowing a controller meant for recovery or backup to take over in the case of a primary controller breakdown. Therefore, for applications to function properly, all controllers requires a stable observation of the network. The centralized solution might also cause scalability issues because every network device must be managed by the same organization. Maintaining a theoretically integrated but actually distributed control plane is one strategy that expands on the concept of employing many controllers over the network. In this scenario, each controller is in charge of controlling a single area of the network, while a common understanding of the entire network is being shared. As a result, although control actions are actually carried out by a distributed system, applications see the controller as a single entity. In addition to eliminating the single failure point, this method has the advantage of improving performance and scalability because each controller component only needs to manage a small portion of the network. Onix and HyperFlow are two known controllers within this category.

The stability in state of the network amidst controller constituents is yet another potential drawback of decentralized control. Since the network's state is spread, it's possible that the applications supported by various controllers have varying perspectives of the network, which could cause them to behave incorrectly. Using two levels of controllers, as the Kandoo controller

does, is a fusion method with focus aimed towards addressing both scalability and stability. A collection of controllers that make up the lowest layer are unaware of the current status of the entire network. Only control activities requiring knowledge of the status of a lone switch are carried out by these controllers (local information only). The top level, on the other hand, is a logically centralized controller in charge of carrying out network-wide actions that call for awareness of the overall network status. The rationale behind this approach is to speed up local activities while minimizing the burden on the high-level central controller and thereby enhancing the network's scalability[42]. In addition to theories about the degree of physical centralization of controllers, theories about their logical decentralization have also been put forth. The Tempest project and the founding years of programmable networks are where the concept of logical decentralization originated. Earlier, we mentioned how the Tempest design allowed for the operation of many simulated ATM networks on uppermost part of a single set of hardware switches. Similar suggestions for SDN proxy controls, such as FlowVisor, that permits many controllers to distribute same dispatching plane, have also been made. The goal of this concept was to make it possible to deploy enterprise and experimental networks simultaneously over the same infrastructure without harming one another. Before we wrap up our assessment of how centralized SDN controllers are, it's critical to look at potential issues with their performance and viability in expansive networking environments. The capacity of SDN networks to balance and remain receptive in situations of huge network traffic is one of the issues brought up most frequently by SDN detractors. This worry is primarily caused by the new paradigm's transfer of control away from network devices and toward a lone organization in charge of overseeing all network traffic[43]. Performance evaluation of SDN controller implementations have been driven

by this worry, and they have shown that physically centralized controllers performs exceptionally well and have incredibly fast response times.

It has been demonstrated that more recent multi-threaded controller implementations perform noticeably better. In a 256-switch network, for example, NOXMT can manage 1.6 million network requests per second with a latency of 2ms on a standard eight-core server with 2.5GHz processing units. Large industrial servers being targeted by newer controller architectures promise to perform much better. For example, the McNettle controller asserts that it can support connections ranging towards 5000 networking devices utilizing a lone 46-core controller having a throughput of more than 14 million requests within a second and a latency of less than 10ms. The placement of controllers within the network is another significant performance issue that is brought up, as network performance tend to be considerably impacted by the number, physical location and techniques used in the coordination of the controllers. The location of the controllers has been viewed as a performance issue, and linkages have been made between this topic and the domains of resident procedures and distributed computing for the development of effective controller coordination protocols[44].

2.3.4.5 Management of Traffic

The management of traffic is another critical design element for SDN controllers. The decisions made about traffic management may directly affect the network's performance, particularly in large networks made up of numerous switches and subject to heavy traffic loads. Control granularity and also policy enforcement are the two areas into which we can subdivide the issues with traffic management.

Control Granularity

The level of control granularity considered in a network traffic describes how acceptable-the controller review actions should compared to the data moving across the system. In traditional setups, each data that arrives the control is independently analyzed, and a transmitting resolution is completed regarding the routing path based on data included (e.g. target address). This strategy typically fit in specifically for traditional setups, SDN is an exception to the rule. The per-packet technique would then be impossible to execute through any sizable setup as every packet are channeled via the controller, which in return create a path specifically designed for them[45]. Large number of SDN controllers adopt a flow-based method because the per-packet approach has performance difficulties. In this technique, every packet is apportioned to a flow based on a specified property. The initial packet that arrives for a fresh flow is examined by the controller, who then configures the switches accordingly.

The enforcement of controller centered on an accumulation flow-match rather than employing discrete flows would be an extra coarse-grained way to further unload the controller. When comparing the level of granularity, the fundamental trade-off is between the burden on the control and service quality provided to network solutions. The per-packet style improves QoS because the control may continuously choose the optimum route for every discrete network packet. Imposing access over an accretion of links results in a partial lack of network state adaptation in the controller's judgments on packet forwarding. In this scenario, packets might be routed over a less-than-ideal route, resulting in reduced QoS.

2.3.4.6 Policy Enforcement

The second problem with traffic management has to do with how the controller applies network policies to network devices. Reactive control models are one strategy used by systems

like Ethane, someplace the interchanging device contacts the control whenever a choice regarding a fresh movement is required. In this instance, network management is more flexible because each flow's policy is defined for the switches only when a real need occurs. Due to the delay in sending the initial request of the flow towards the controller for review, this method could potentially result in performance reduction. In circumstances when controllers are positioned in far location from the switch, this performance reduction could be severe. Using a proactive control model would be an alternate strategy for enforcing rules. In this state, the controller pre-populates the routing boards with some traffic which might pass over the gadgets before pushing the procedures to every network switches. By using this method, a switch can create a new flow without asking the controller for instructions and instead by performing a check at the table previously deposited in the device. The benefit of proactive mechanism infers it does away with the delay caused by asking the controller about each movement.

2.3.5 SDN Programming Interfaces

Further detailed discussion about the main ideas and problems with SDN programming are emphasized by looking through each communication point on its own.

2.3.5.1 Southbound Communication

The southbound communication is crucial for the controller's ability to influence the behavior of SDN switches. This is an example of how SDN tries to "program" the system. OpenFlow remains a well-known illustration of a standardized northbound API. Since OpenFlow-based interaction amongst the controller and switches is assumed in the majority of SDN-related projects, it is crucial to present OpenFlow in great detail[46].

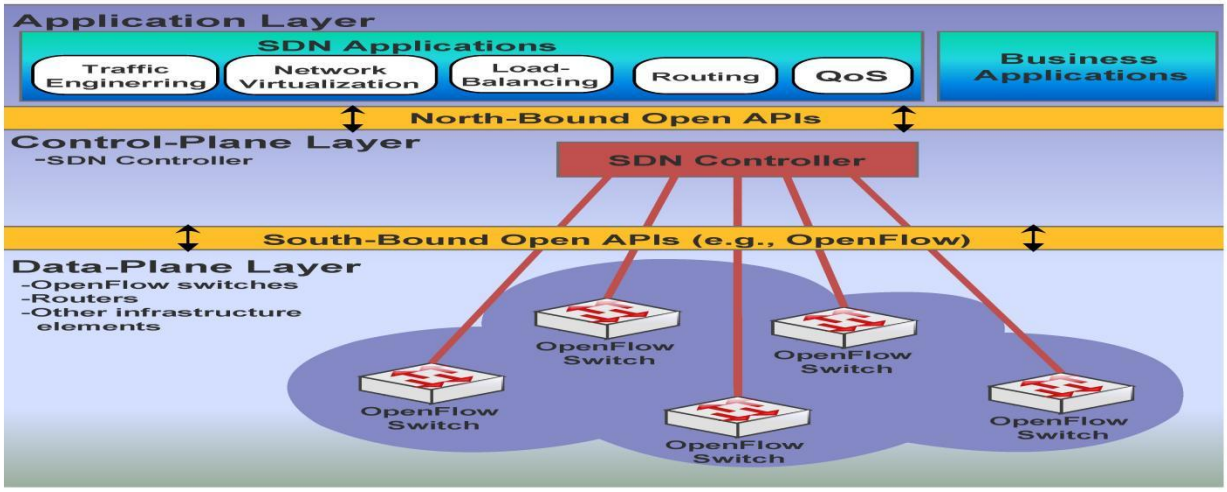

Figure 2.15: SDN Southbound Communication

(Source: https://www.mdpi.com/2079-9292/9/7/1091/htm)

It should be made obvious that there are other different implementations of controller-switch interactions, with OpenFlow being just one (although a quite popular) of them. There are more solutions, such as DevoFlow, that aim to address OpenFlow's performance problems.

2.3.5.2 Overview of OpenFlow

As shown in Figure 2.16, OpenFlow offers a consistent method of handling congestion in devices and of communicating data amid the switches and the controller, adhering to the SDN standard of separating the control and data planes. Two logical components make up the OpenFlow switch. One or more movement chart are present in the first component and are in charge of keeping the data needed by the switch to transmit packets. The other part is an OpenFlow user, which is a straightforward API that enables message between the controller and switch.

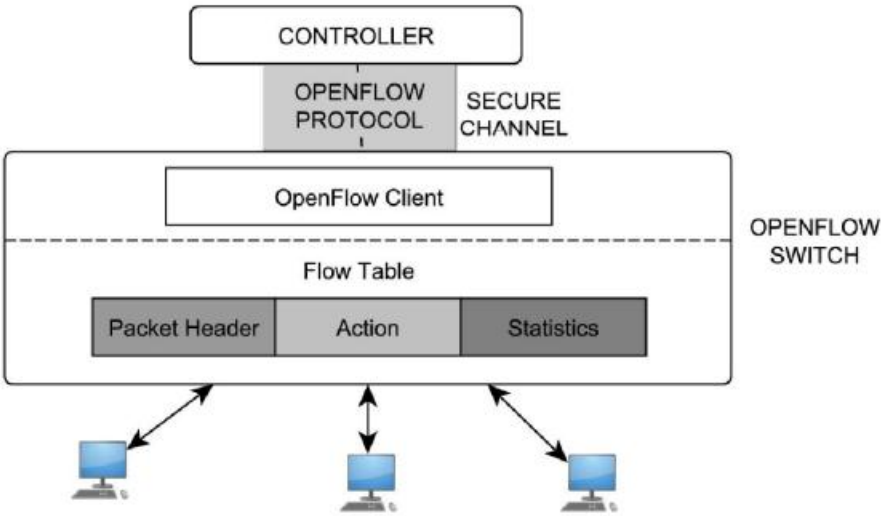

Figure 2.16: OpenFlow switch and communication with the controller

(Source: https://www.mdpi.com/2079-9292/9/7/1091/htm)

The flow tables are made up of movement accesses, individual of which specifies a group of guidelines for what way the switch will handle request associated with that specific movement. The movement chart contains three parameters for every entry: three components: (a) a packet caption identifying the movement; (b) an action stipulating how the data would be managed; and (c) indicators, which retain path of data like the quantity of data and bytes in individual movement as well as the duration as a data belonging to a movement was most recently delivered. When a request reaches the OpenFlow gadget, its caption is analyzed, then the flow with the most comparable packet header field is used to match the packet to the movement. The action indicated in the action aspect is carried out if a matching flow is discovered. These comprise conveying the request to a certain location so that it can be routed via the network, sending the packet so that the controller can review it, or rejecting the packet. If no flow can be found to

match the packet to, it is handled in accordance with the procedure specified in a table-miss movement entrance[47].

Transferring communications across a protected network in a standardized manner outlined by the OpenFlow procedure is how information is exchanged amidst the switching gadgets and the controller. Following this approach, the control can modify the flows listed in the switch's flow table—for example, by adding, updating, or deleting a flow access—either pro-actively or reactively, such as explained in the fundamental control concepts. Network operators are no longer required to interface directly with the switch because the control may communicate with the device using the OpenFlow model. The fact that the packet header field in OpenFlow can be a wildcard, allowing for more flexible matching to packet headers, is one of its most enticing features. This method is predicated on the notion that different network devices share a common dispatching conduct and simply differ in the caption data being utilized for identifying and the steps taken. OpenFlow conceptually unites a wide variety of network device types since it permits the usage of any subclass of these caption fields for relating procedures to movement flows[48].

2.3.5.3 Northbound API

The availability of a network OS, which sits amidst the network setup as well as the highly ranked facilities and requests, much like a computer OS does, is one of the fundamental concepts favored by the SDN paradigm, as was already mentioned. A precise interface for the interaction amongst the controller and the applications ought to likewise be present in the SDN design, supposing such a compacted coordinating unit and centered on the fundamental OS modalities.

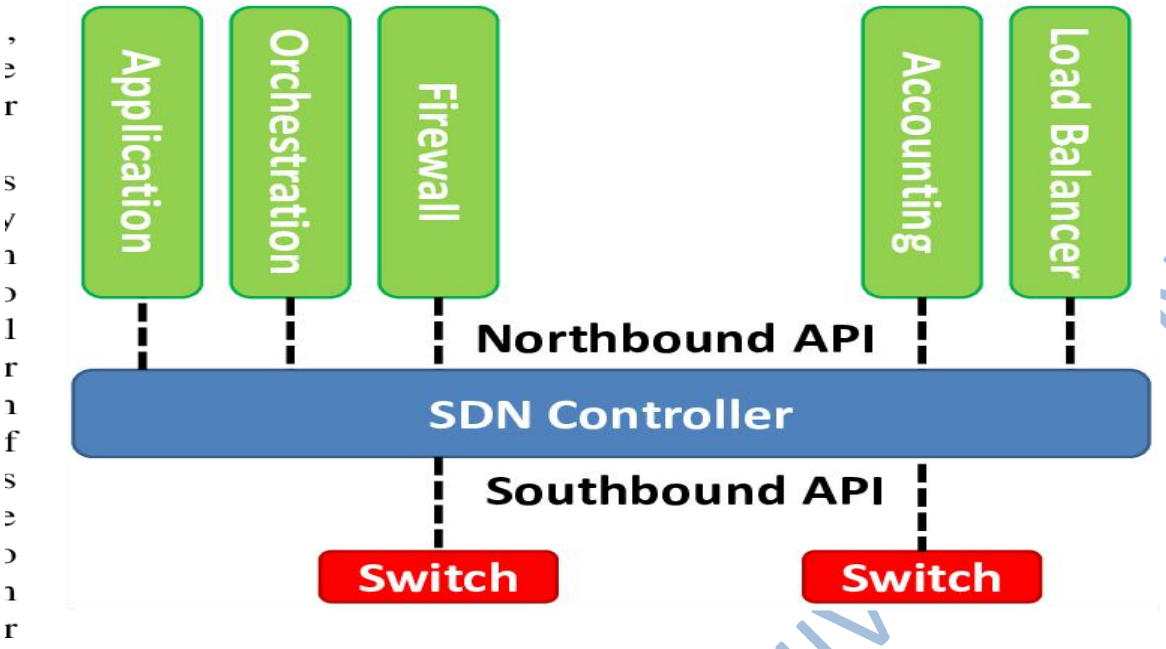

Figure 2.17: Northbound and Southbound API

(Source: Source: https://www.researchgate.net/figure/The-Southbound-and-Northbound-SDN)

Through this interface, programs should be able to interact with one another, manage system resources, and access the underlying hardware without needing to be aware of low-level network information. There is now no accepted procedure for interfacing the controller with the applications, in divergence to its southbound counterpart, where communications between the switches and the control are clearly specified by means of a specified common[49]. As a result, each controller model must offer unique means of carrying out controller-application communication.

It is also challenging to design applications with various, frequently conflicting aims that exist as a result of other complex models because even the boundaries that existing controllers generates offer extremely low-level concepts (e.g., movement handling). Consider the applications for power control and a firewall as examples. Although the firewall may require

some additional controls to direct circulation so that it utmost complies with all security regulations, the power management program must redirect traffic using fewer links in order to deactivate idle switches. It may become a very difficult and time-consuming process to let the coder handle these disputes. The usage of complex network software design languages that translate rules into low-level flow limitations has been advocated as one solution to this issue. The controller would then use these languages to govern the SDN switches. Within SDN architecture, these network software design languages can similarly be thought of as per an intermediary level that sits between the application layer and the controller, much too by what means the complex scripting languages like C++ and Python are built on top of low level language to shield programmers from the assembly language's intricate low-level intricacies. Frenetic and Pyretic are a couple of illustrations of such complex network software design languages.

2.3.6 SDN Application Domains

Discussions below best describe two defining situations in which SDN have proven to be advantageous: this includes in data centers and cellular networks, so as to show the SDN can be applied in a variety of networking disciplines. The number of ways SDN can be applied is, of course, not just restricted to a single areas but also includes a wide range of others.

2.3.6.1 Data Center Networks

Finding scalable solutions towards sustenance of thousands of connected servers and countless number of networked virtual mechanisms is among the crucial criteria for data centered interconnection. From a network standpoint, obtaining such scalability can be difficult. First of all, as servers are added, forwarding tables grow in size, necessitating the use of more

advanced and expensive forwarding devices. Moreover, given that datacenters are required to consistently accomplish high levels of performance, traffic management and policy implementation can become highly significant and vital challenges. The underlying network in traditional datacenters is carefully designed and configured in order to gratify the aforementioned requests. Most time, this activity is carried out manually by establishing the desired traffic paths and positioning intermediate boxes at key physical network choke points. Since manual setting can be difficult and error-prone, especially as the network gets bigger, this approach obviously goes against the need for scalability.

Since the data center cannot constantly adjust to the application requirements, it also gets harder to get it to operate at maximum capacity. These holes are filled by the benefits that SDN provides to system administration. The control layer is separated from the data layer, making forwarding devices far more straightforward and less expensive. In addition, one logically centralized entity is given power over all control logic. This enables dynamic flow management, load balancing of traffic, and resource distribution in such a way that best supports the task of the data warehouse with the requirements of operating solutions, and as such results in improved performance[50]. Finally, since strategy application are now accomplished via the controller object, adding middle boxes to the network is no longer necessary.

2.3.6.2 Cellular Networks

Cellular links may be regarded as parts of the broadcastings market. Over the past ten years, the quantity of cellular devices has grown quickly, straining the capabilities of the current cellular networks. SDN principles have recently attracted a lot of interest for cellular designs now in use. One of the key shortcomings of existing cellular network designs is that the

network's main has an integrated data movement. All movement must travel over dedicated tools that performs several system tasks, such as access control, billing, and routing (for example, packet gateway in LTE). This complexity of the devices drives up the cost of the infrastructure and poses serious scalability issues. To meet the frequent request from the always growing movement and the constrained wireless band for network access, the access system's cell sizes also have a tendency to shrink. However, this causes more interference between nearby base stations and causes the load to fluctuate from one base station to another as a result of user mobility, making the static distribution of resources insufficient. Some of these issues could be resolved by cellular networks implementing the SDN principles. First off, the introduction of an integrated control with a full observation of the full system and the disintegration of the control from the data plane allow for the simplification of network equipment, which lowers the cost of the overall infrastructure.

Additionally, because different collaborating controllers can be assigned to carry out certain tasks like channeling, concurrent observation and feedback, movement controlling, entrance control, as well as strategy implementation, the network is more flexible and simple to maintain. Furthermore, by eliminating the need for base station cooperation and direct communication, having a central controller that serves as an abstract base station makes load management and interference management operations simpler. Instead, the controller simply gives the data level operational instructions and makes decisions for the entire network[51]. One other benefit is that the adoption of SDN makes it easier for virtual operators to enter the telecom sector, increasing competition. By virtualizing the underlying switching hardware, all providers can manage their own subscribers' flows through their own controllers without having to fork over a lot of money to build out their individual setup.

2.3.7 Relation of SDN to Network Virtualization and NFV

Network virtualization and Network Functions Virtualization are two extremely popular SDN-related technologies (NFV). Since these technologies frequently cause misunderstandings, particularly among people who are unfamiliar with the notion of SDN, we make a brief effort to explain their link to SDN in this paragraph. Network distribution and the core physical framework are separated by network virtualization. As a result of virtualization, the possibility of having numerous "virtual" systems running through similar physical hardware, they are identified to have a layout that is significantly less complex than the real network[52]. With the help of this abstraction, network administrators can build networks according to their needs without having to make changes to the original substructure, this which have proven to be challenging or even not possible.

The idea of system virtualization, which remains to isolate the network from the underlying physical structure, is similar to the idea of SDN, which is to detach the regulator from the information plane, and this naturally leads to confusion. The two technologies are independent of one another, in actuality. Network virtualization is not automatically implied by the presence of SDN. SDN is not always required to achieve network virtualization, either. On the other hand, an SDN system may be put in a virtualized setting while a network virtualization template could be deployed on an SDN setup. Network virtualization served as one of and arguably the utmost significant use cases of SDN, and the two technologies have coexisted closely ever since. The explanation is that SDN's architectural flexibility made network

virtualization possible. In other words, SDN is one (perhaps the best) architecture for accomplishing this, while network virtualization are considered as a way out concentrating on a specific issue. However, as was previously emphasized, network virtualization must be viewed separately from SDN. In fact, many have claimed that network virtualization may end up being an even greater technological advance than SDN. Network Functions Virtualization is a separate but closely related technology to SDN (NFV). By using virtualization-related tools to project network operations like intrusion discovery, caching so they may operate in solutions, NFV is a carrier-driven project with the aim of changing how operators construct networks[53].

Finally, because it is feasible to swiftly adapt or offer new services to meet shifting demands, network administration becomes more flexible. The separation of the two technologies can be a little hazy because the separation of system operations from the core hardware is strictly linked towards the separation of the regulating plane from the data plane proposed in SDN. Although they are closely similar, SDN and NFV pertain to dissimilar purviews, which is a crucial distinction to make. SDN and NFV are complementary but independent of one another. For instance, based on NFV technology, the regulating purposes of SDN might be realized as virtual roles. The forwarding actions of physical switches, however, might be managed by an NFV orchestration system using SDN. However, neither technology is necessary for the other to function; rather, both could profit from the advantages that each technology has to offer.

2.3.8 Effect of SDN to Research and Industry

The impact of SDN on the scientific community and the corporate world will now be discussed briefly. While each interested party may have a different focus, from creating innovative solutions that take usage of SDN towards creating SDN-enabled solutions suited for

commercial contexts, their participation within the development of SDN aids to shape the prospect of this skill. We can get a sense of what could hypothetically motivate further study in this area by looking at the inspiration and concentration of existing SDN-related endeavors.

2.3.8.1 Outline of Standardization Activities and SDN Summits

In recent times, a number of normalization groups have been concentrating on SDN, individually attempting to provide homogeneous elucidations for a specific area of the SDN domain. Since standardization is the first stage towards a technology's widespread acceptance, the advantages of such initiatives are enormous. The Open Networking Foundation, a non-profit industry collaboration, is thought to be the most important standardization group for SDN. In addition to telecommunication operatives, network and facility suppliers, equipment purveyors, and providers of interacting and virtualization solutions, it has more than 100 corporate members. Through the use of open SDN standards, it seeks to convert the networking sector into a software sector, making SDN the new standard for networks.

It works to normalize and publicize SDN and the expertise that support it, with the normalization of the OpenFlow procedure being its key success. The ONF maintains a variety of working groups that focus on various facets of SDN, including abstractions for forwarding, extensibility, configuration, and management, as well as educating the public about the benefits of SDN[54]. Finally, it makes an effort to pinpoint SDN usage scenarios and potential research roadblocks. The Boundary to the Transmitting System operational group is creating an SDN approach as an alternative to OpenFlow, which allows for the operation of conventional dispersed routing protocols on network hardware in order to transmit data to a centralized

management. Other SDN-related IETF operational groups include CDNI, which is researching how SDN might be aimed at Content Delivery Network (CDN) interconnection, and ALTO, which uses SDN to optimize application layer traffic. ITU-Telecommunication T's Standardization Sector study groups (SGs) are looking at SDN for unrestricted telecommunication systems.

As a final objective, MEF seeks to create, advance, and accredit technical standards for carrier Ethernet services. Examining if MEF services would operate within the ONF SDN framework is one of its directions. In addition to the effort being done to standardize SDN clarifications, there are an amount of conferences for exchanging and examining fresh concepts and significant advancements within in the SDN[55].

2.3.8.2 SDN within the Industry

The benefits that SDN proffers over customary networking have also caused the networking community to concentrate on SDN, aimed at creating and supplying commercial SDN solutions or for adopting it as a way to use it to abridge administration also further advance facilities within their secluded systems. The rapid expansion of Google's back-end grid, according to Google engineers, was the primary driver for switching to the SDN paradigm. As scale grows, computational and storage costs decrease, but the network does not follow suit. Applying SDN principles allowed the business to pick networking gear with the functionality it needed while also creating ground-breaking software.

Additionally, the integrated system control increased the network's efficiency and responsibility tolerance, fostering a further adaptable and creative atmosphere even though also lowering operational costs. In recent time, Google unveiled Andromeda, a version of its software

defined network that powers the company's cloud and aims to improve the scalability, affordability, and speed of Google's services. Amazon and Facebook, two other significant networking and cloud service providers, intend to base their future network infrastructure implementation on SDN concepts. Additionally beginning to express interest in creating commercial SDN solutions are networking companies. There is a tendency toward building whole SDN ecosystems aimed at various client categories rather than just developing certain solutions like OpenFlow controls and network OS[56]. In the SDN space, for example, firms like Cisco have introduced their own comprehensive solutions aimed at businesses and cloud service deployment agents, while telecom firms like Huawei focuses on developing resolutions aimed at the forth-coming age of telecommunication grids.

2.4 Load Balancing Technics

Distributing network traffic among several servers is a practice known as load balancing. By doing this, it is made sure that no server is overloaded. Load balancing increases the responsiveness of an application by distributing the work evenly. Additionally, it makes more websites and applications accessible to users. Without load balancers, modern apps are unable to function. Software load balancers have expanded their capabilities over time, including features like application security[57]. A server farm's usage of load balancing as a fundamental networking method for traffic distribution amongst several servers. Load balancers enhance the responsiveness and availability of applications while preventing server overload. Each load balancer lies in between client devices and the backend servers, taking incoming requests and then distributing them to any available server that can fulfill them.

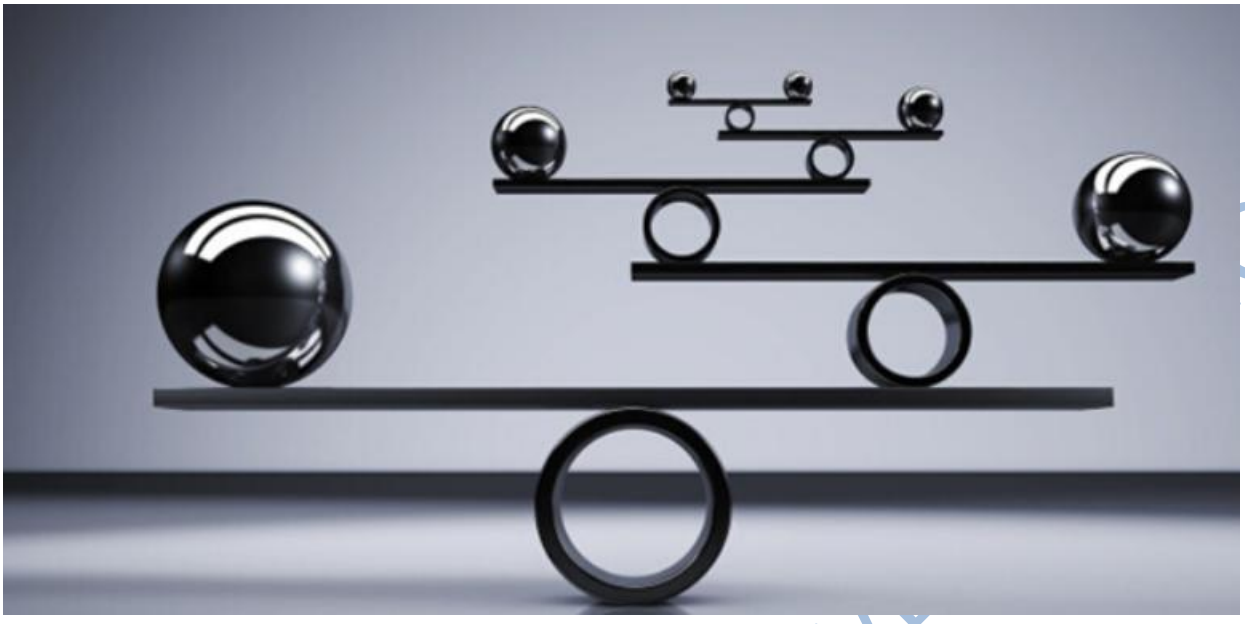

Figure 2.18: Load Balancing

(Source: Source: https://google.com)

The process of effectively dispersing incoming network traffic among a collection of backend servers, commonly referred to as a server farm or server pool, is known as load balancing. Modern high-traffic websites must quickly and reliably respond to hundreds of thousands, if not millions, of concurrent user or client requests for the right text, photos, videos, or application data. Modern computing best practice typically necessitates the addition of extra servers in order to cost-effectively scale to handle these enormous volumes[58]. A load balancer serves as the "traffic cop" in obverse of the servers, distributing incoming demands through all servers equipped to handle them in a way that exploits speed and ability consumption and makes sure that no server is overused, which can lead to performance degradation. The load balancer routes request to the responsive servers in case one server goes offline. The load balancer initiates requests to a new server when it is added to the server group.

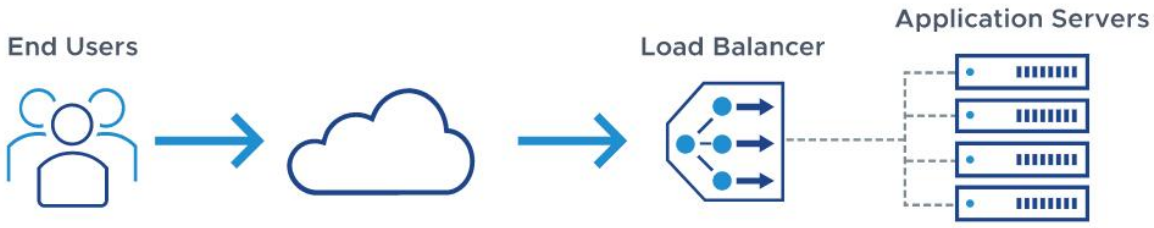

Figure 2.19: Client-Server Connection with a Load Balancer

(Source: https://avinetworks.com/what-is-load-balancing/)

The following tasks are carried out by a load balancer:

- ✓ Effectively distributes client requests or network strain among several servers
- ✓ By routing requests only to servers that are fully operational, it ensures excellent availability and reliability.
- ✓ Gives the option to increase or reduce the quantity of servers according to request

2.4.1 Load Balancing Techniques

A distributed network is primarily heterogeneous in nature because distinct networks, which are dispersed far throughout the globe, may have different processing nodes, network topologies, communication channels, operating systems, etc[59]. The distributed computing system is currently constructed using connections between several hundred PCs. The overall work load must be dispersed among the nodes within the network in order to achieve a system's optimal efficiency. Therefore, the presence of distributed memory multiprocessor computer systems led to the issue of load balancing becoming well-known. The load balancing problem is the general term for the circulation of requests to the processing units. There is a very high likelihood that certain nodes in a system with several nodes will be indolent whereas the others will be

overloaded[60]. The purpose of the load distribution procedures is to keep the load on each handling unit in a way that none of the processing essentials turn out to be overloaded or idle. To achieve the system's optimal performance (lowest execution time), individually processing element should preferably have an equivalent weight at any given point throughout execution. Therefore, a load balancing algorithm that is properly designed could greatly enhance the system's performance.

Both fast and slow nodes with regard to computation will be present in the network. If processing speed and connection speed (bandwidth) are not taken into consideration, the network's slowest functioning node will have an impact on the system's overall performance. Therefore, load balancing solutions maintain a balance between the requests on the nodes by preventing certain nodes from being idle while the others are overloaded[61]. Additionally, load balancing techniques eliminate any node's inactivity at runtime. Two things affect how well a load balancing algorithm performs. First, the steps that must be taken in order to achieve equilibrium. The second factor is the amount of load that travels over the link that connects the nodes.

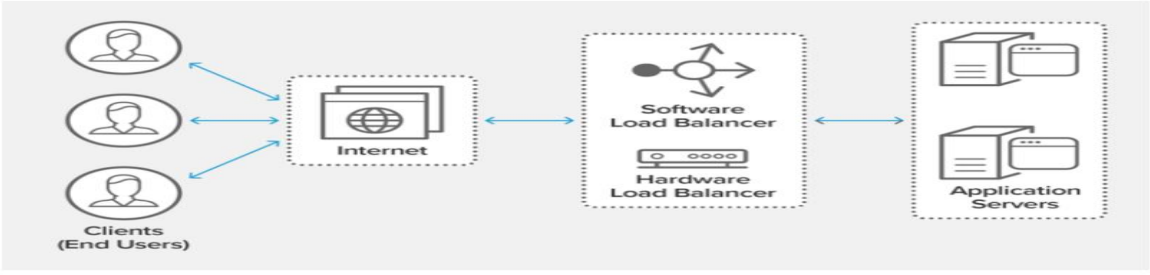

Figure 2.20: Client-Server Connection with a Load Balancer

(Source: https://www.nginx.com/resources/glossary/load-balancing/)

Load balancing is a technique for spreading task units among a group of interconnected processors that may be dispersed throughout the world. Other processors with loads below the threshold load get the extra or unfinished load from one processor. A processor's inception jobs is the maximum quantity of load it can handle before any additional load is applied. There is a very high likelihood that certain nodes in a system with several nodes remains unused whereas the others will be overloaded. Therefore, the processors in an arrangement can be classified as strongly loaded (sufficient tasks are ready for completing), lightly weighed down (fewer works are waiting), and idle processors based on their current load (have no job to execute). It is likely to make each processor evenly occupied and to accomplish the tasks roughly at the same time by using a load balancing approach.

There are three rules that make up a load balancing operation. These are the distribution rule, location rule, and selection rule. The selection rule operates in a non-preemptive or preemptive manner. A non-preemptive rule is always applied to freshly formed processes, but a preemptive rule may be applied to active processes. Preemptive transfers are more expensive than non-preemptive transfers, which are the better option. Preemptive transfer, however, can occasionally be superior to non-preemptive transfer[62]. Practically, location and distribution rules work together to decide on load balancing. There are two categories of balancing domains: local and global. Within the local purview, the balancing resolution is made by the closest neighbors by sharing local workload data, whereas in the global domain, transfer partners are activated throughout the entire system and work load data is exchanged internationally.

Although numerous load balancing methods have been created over the past few years, no single technique is suitable for all situations. The choice of a suitable load balancing depends on hardware features like communication overheads as well as application parameters like

balancing quality and load generating patterns[63]. In general, there are dual kinds of load distribution processes: static weight distribution and dynamic weight distribution. The document and index data are divided through closely-clustered disseminated computing arrangements in dispersed structures to accommodate the required data quantities and query throughput rates. Uneven load distributions that are common in real-world applications lead distributed systems to perform poorly and scale poorly. The lack of a load balancing solution is the cause of this.

The HTTP traffic must be divided equally among the web servers in a server group when there are several web servers present. These servers must look to the web devices, such as an internet browser, as a single web server during the process. By evenly distributing workload among dual or additional processors, network links, CPUs, hard drives, or other resources, load balancing achieves the best possible resource usage, throughput, reaction time, and overload prevention. Through redundancy, using numerous components with load balancing as opposed to a single component may boost reliability. Typically, a specialized software application or piece of hardware delivers the load balancing service (such as a multilayer switch or a DNS server). To maximize resource efficiency in Data Grids, load balancing and scheduling are crucial. The field of load balancing has seen a lot of research. For instance, numerous load balancing policies have been developed to increase efficiency by better utilizing CPU or memory resources. Since the majority of existing load balancing solutions overlook the issue of load-balancing among storage resources, they are insufficient for Data Grids supporting data-intensive applications[64]. The load balance problem has been solved using numerous algorithms and studies. Each of these research illustrates how we might assess the effectiveness of each algorithm that has been offered. Some of these studies assess these algorithms based on particular criteria, including performance analysis of load balancing methods.

2.4.2 Types of Load Balancers – Based on Functions

There are several load balancing methods available to deal with the distinct network issues:

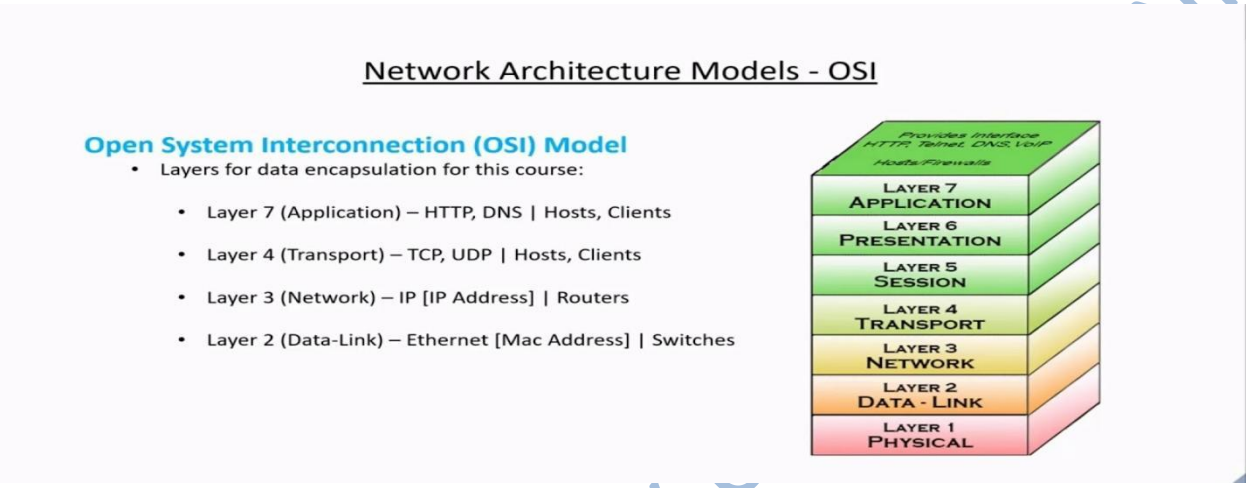

Figure 2.21: OSI-model

(Source: https://electricalfundablog.com/osi-model/)

a.) Network Load Balancer / Layer 4 (L4) Load Balancer:

Network weight manager refers to the distribution of load at the transport layer via the transmitting resolutions, this is done centered on link elements which includes IP address and target ports. This type of weight harmonizing is TCP-based, or level four, and it disregards all application-stage factors for instance content kind, cookie information, headings, positions, application behavior, etc. Network load balancing simply considers the information at the network layer and only uses this information to perform network addressing translations without examining the contents of individual packets[65].

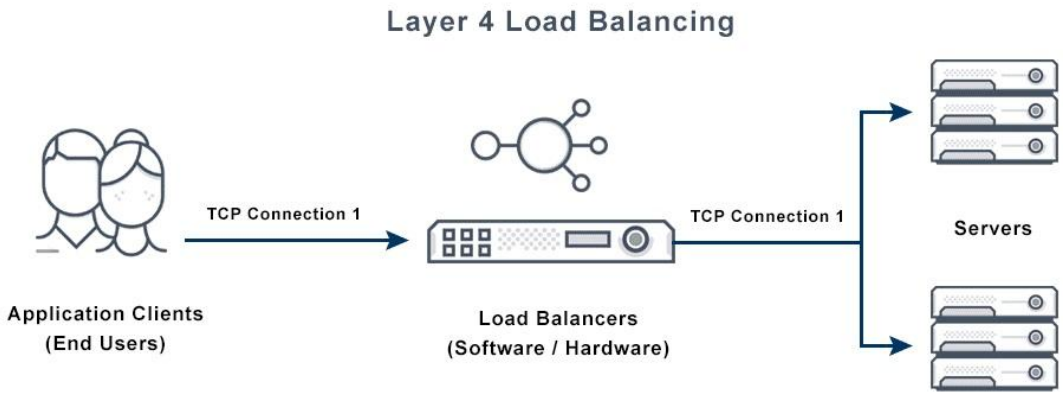

Figure 2.22: Layer 4 Load Balancer

(Source: https://www.nginx.com/resources/glossary/load-balancing/)

b.) Application Load Balancer / Layer 7 (L7) Load Balancer:

The Layer 7 load balancer, which sits at the top of the OSI model, distributes the requests based on several application-stage factors. The L7 weight stabilizer analyzes a considerably wider assortment of facts, together with HTTP captions and SSL periods, and allocates the server weight centered on the choice made by the interaction of numerous factors. Application load balancers can then manage server traffic based on user usage and behavior in this way.

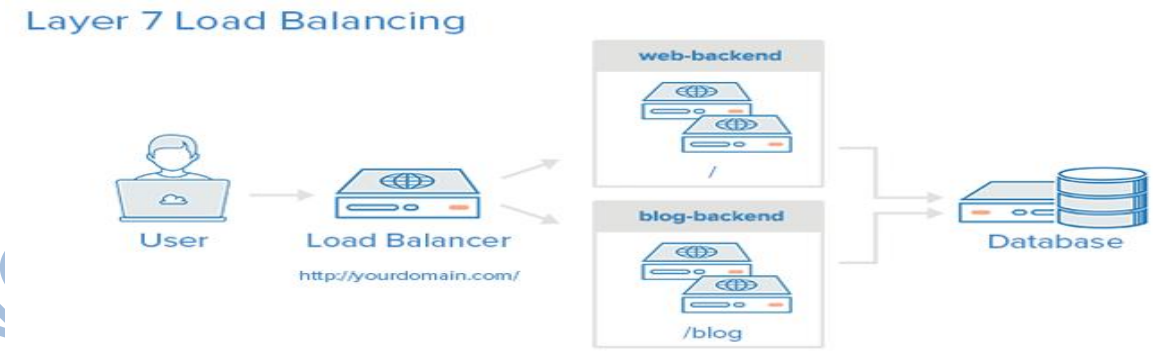

Figure 2.23: Layer 7 Load Balancer

(Source: https://www.nginx.com/resources/glossary/load-balancing/)

c.) Global Server Load Balancer/Multi-site Load Balancer:

The GSLB expands the competencies of general Layer four and Layer seven across numerous information hubs, expediting the effectual universal load dissemination devoid of diminishing the experience for finale users[66]. This is necessary because an accumulating quantity of requests are being housed in cloud data hubs, which are dispersed across various geographies. Other than effectively balancing movement, multi-site load distributions also aid in speedy salvage and flawless company processes in the event of a server tragedy at any data hub since they provide business continuity by allowing usage of other data centers anywhere in the world.

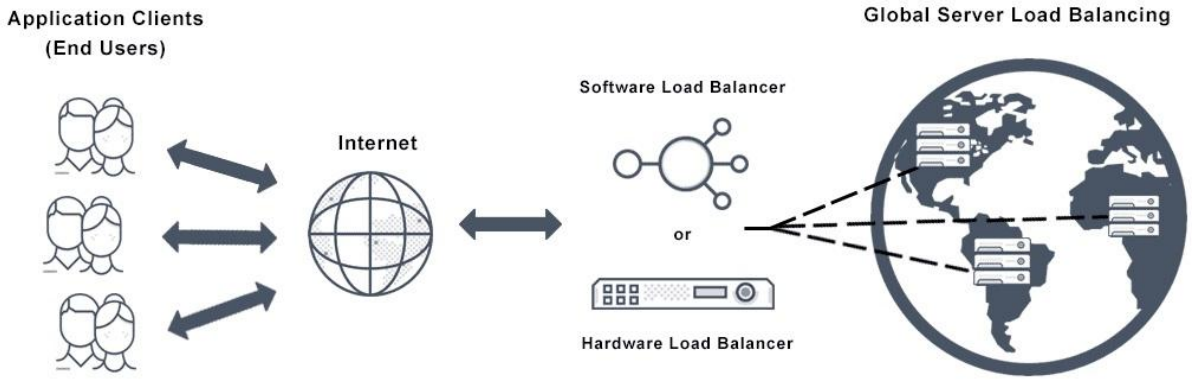

Figure 2.24: Global Server Load Balancer

(Source: https://www.nginx.com/resources/glossary/load-balancing/)

2.4.3 Types of Load Balancers – Based on Configurations

Load Balancers are also classified as:

a.) Hardware Load Balancers:

This is a physical piece of on-site hardware, as the name would imply, used to distribute traffic among several servers. Despite having a large amount of traffic handling capacity, they have a limited amount of flexibility and are also extremely expensive. A hardware load balancer distributes web application traffic among a group of application servers using a specific operating system. The hardware load balancer distributes traffic in accordance with specific rules to provide optimal performance while preventing application servers from becoming overloaded. In on-site data centers, hardware load balancers and application servers are typically placed, with the number of load balancers used based on anticipated peak traffic volumes. In the event that one fails, load balancers are typically deployed in pairs.

Figure 2.25: Hardware Load Balancer

(Source: https://www.loadbalancer.org/products/hardware/enterprise-1g/)

A hardware load balancer acts as a "traffic cop" by sitting between incoming traffic and internal servers. When users access the website, the load balancer receives their request first and then routes them to various servers. Global server load balancing is a practice used by the majority of businesses that involves the deployment of load balancers and servers in several

locations (GSLB). With GSLB, high availability is guaranteed during disaster recovery scenarios in addition to optimized response times. By rerouting network traffic to other accessible sites in the event of a data center failure, GSLB systems can lessen the impact on end users.

b.) Software Load Balancers:

They remain software programs that must be mounted on the structure and work in a manner similar to hardware load balancers. A distributed data level and a unified control layer make up the architecture upon which software defined load balancing is based. The services provided by the data layer are organized by the control layer. To decide on service placement, auto scaling, and high availability for each application, it receives and analyzes the constant stream of application telemetry supplied by the distributed load balancers across the environments. Software defined load balancers can be set up and operated by the control plane in a variety of settings, including data centers and public clouds.

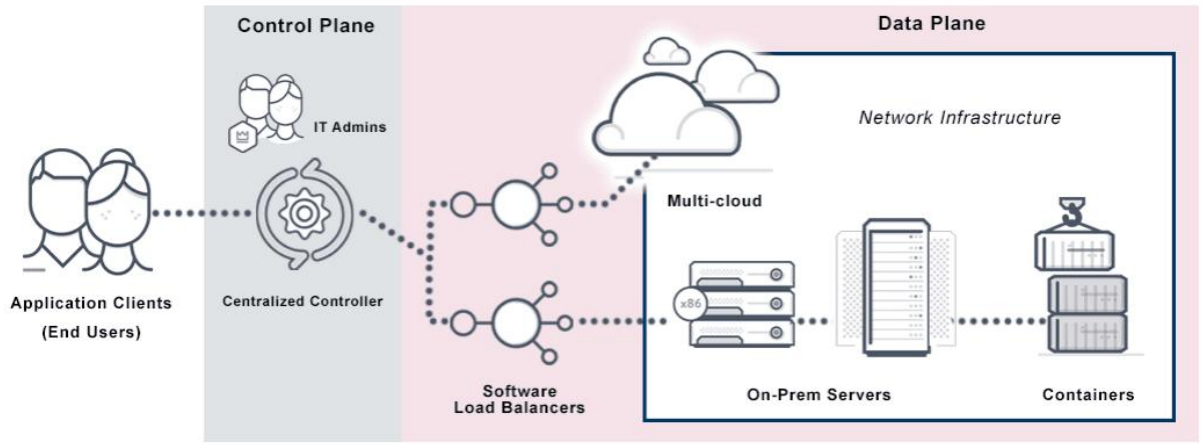

Figure 2.26: Software Load Balancer

(Source: https://www.semanticscholar.org/paper/)

A software-defined load balancer, which does not take the form of a physical appliance, is placed in front of servers and is responsible for directing traffic. The load balancing software distributes client requests among all servers in order to maximize speed and usage and avoid performance degradation by making sure that no server is overworked. If one server fails, the software-defined load balancer reroutes traffic, spreading requests to the remaining online servers. Additionally, it automatically sends requests to any additional servers that are included in the server group. To put it another way, software defined load balancing effectively distributes client requests and network traffic among all suitable servers. The administrator has the option to add and remove servers as necessary. Additionally, by only sending traffic to servers that are available, it guarantees excellent availability and reliability. By removing protocols at the hardware level, load distribution for software based interconnection enables better diagnosis and network administration[67]. Software oriented interconnection load distribution does not rely on the algorithms created for conventional load balancers, even though they are used by traditional network equipment to define such algorithms[68].

c.) Virtual Load Balancers:

This job distributor differs from either software or hardware load distributors because it combines a hardware load balancer's software with a virtual device. This form of load equalizer mimics the software-focused structure through virtualization. In order to get the traffic directed appropriately, the hardware equipment's software application is run in a virtual machine. However, these load balancers face the same difficulties as the physical on-premises balancers, namely a lack of central control, a lack of scalability, and a very low level of automation[69]. By dividing traffic among numerous network servers, a virtual task equalizer offers additional litheness in harmonizing a server's encumbrance. Through virtualization, virtual task pairing

seeks to resemble software-focused structure. On a virtual machine, it runs the load balancing appliance's software.

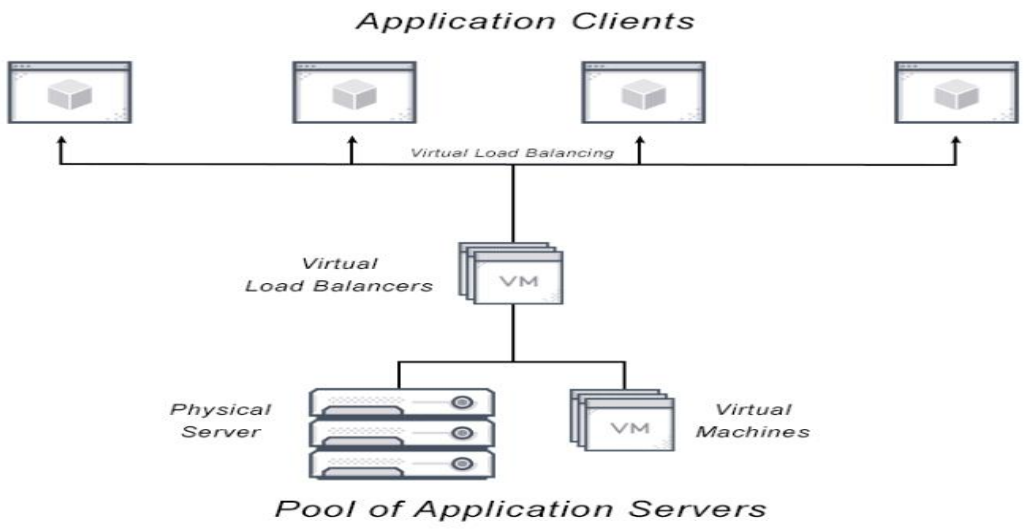

Figure 2.27: Virtual Load Balancer

(Source: https://avinetworks.com/glossary/server-load-balancer/)

A virtual network load balancer makes the claim that it can provide software load balancing by executing the software of a physical appliance on a virtual machine load balancer. However, virtual load balancers are only a temporary fix. Traditional hardware appliances still face architectural issues including restricted automation and scalability, as well as a lack of central management (including the disintegration of control layer and data layer in data centers).

2.4.4 Load Balancing Pairing

The pairing demands are delivered to various kinds of load equalizer, and they are then handled in accordance with a predefined algorithm. Incoming server requests or traffic are efficiently distributed across the servers in the server pool using load balancing algorithms or processes. In order to guarantee Web services' high availability and their quick and dependable

delivery, efficient load balancing is required. Servers are replicated in order to handle a large traffic demand. Task harmonizing is the technique of allocating an incoming job or request to a server among such duplicated servers. Several load balancing techniques, like as round robin, least connections, adaptive balancing, etc., are used to successfully schedule the routing of requests from a client to the appropriate servers in an optimum way.

2.4.5 Survey of Few Existing Load Balancing Algorithms

For task harmonizing and/or maximizing the utilization of the entire servers in distributed structure, load balancers employ various traffic management methods. Algorithms enable distributed systems to operate at higher bandwidths and with faster reaction times. Each algorithm has strengths and weaknesses.

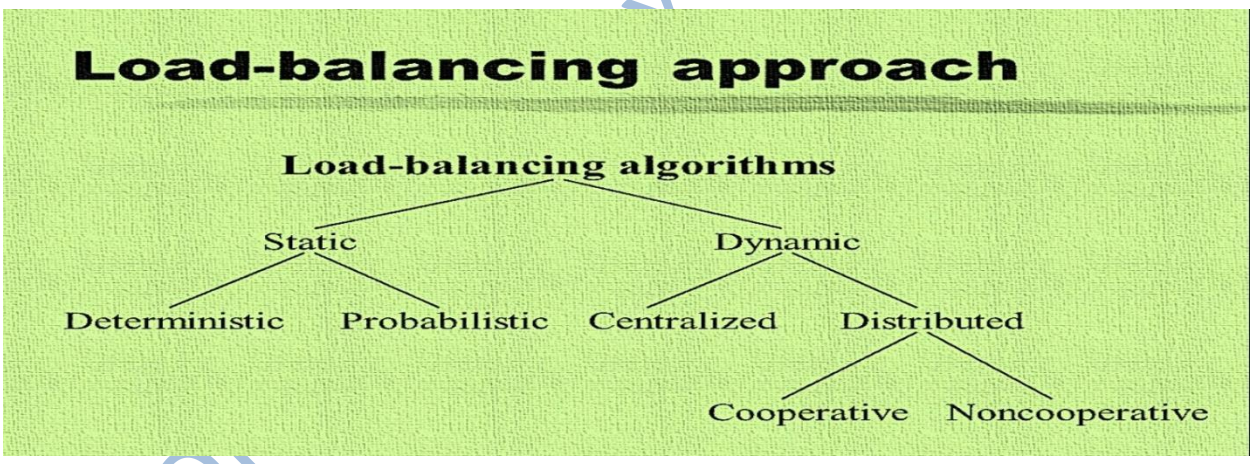

Figure 2.28: Load Balancing Approach

(Source: https://google.com)

Static Load Balancing Algorithm: techniques distribute traffic without taking into account the servers' or system's present condition. Some static algorithms distribute the traffic equally across the servers in a group, either in a predetermined order or at random. When distributing jobs, a

fixed task harmonizing procedure does not take the system's state into consideration. Instead, prior knowledge and assumptions about the entire system are used to shape distribution.

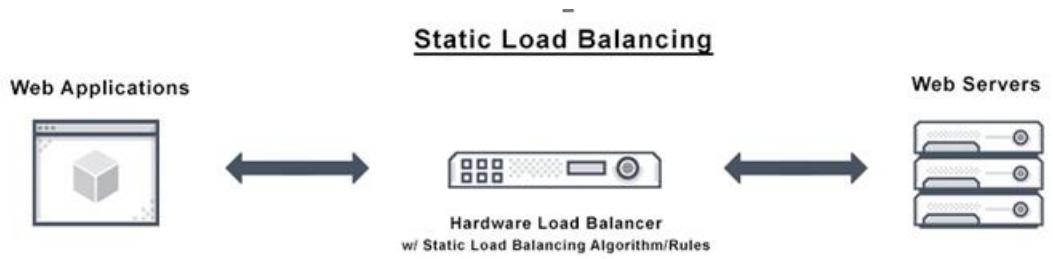

Figure 2.29: Static Load Balancing Rules

(Source: https://google.com)

This comprises both knowns—like the number of processors, the communication speeds, and the power—and assumptions—like the quantity of resources needed, the speed of responses, and the arrival timings of incoming tasks. Distributed systems with static load balancing techniques reduce some performance functions by matching a predetermined set of jobs with available processors. These load balancing techniques frequently center on a router that balances loads and improves performance. Static load balancing in distributed systems has the advantage of being simple to use and deploy quickly, yet there are some cases when it is not the optimal algorithm to apply.

Dynamic Load Balancing: technique takes into thought the present workload of each node or computing unit in the system, allowing jobs to be dynamically moved from weighed down nodes to less loaded nodes to speed up processing[70]. Dynamic algorithms are far more difficult to design, but they can deliver better results, particularly when the execution times for several jobs

vary substantially. Furthermore, a dynamic load balancing design is frequently more modular because it is not necessary to dedicate certain nodes to job distribution.

Figure 2.30: Dynamic Load Balancing Rules

(Source: https://google.com)

Unique assignment of tasks entails the assignment of tasks to a processor based on its condition at a given time. Dynamic assignment refers to the persistent reassignment of duties based on the system's current condition and its evolution. Evidently, any load balancing technique might actually make the entire process take longer if it necessitates a lot of communication to reach a conclusion.

2.4.6 Load Balancing Methods

Load balancing methods are often referred to as algorithms for load balancing or scheduling methods since they explain how a server's load is distributed across a server pool. There are several load balancing strategies that can be used, and each strategy schedules incoming traffic according to a different criterion. The following are some of the typical load balancing techniques:

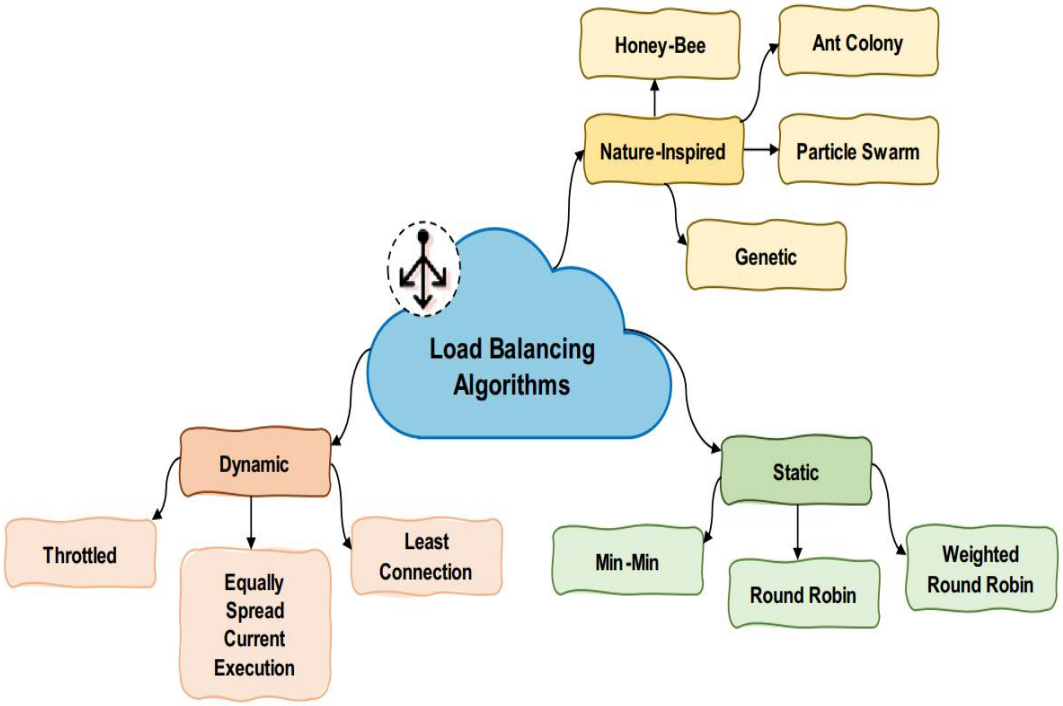

Figure 2.31: Taxonomy of Load Balancing Algorithms

(Source: https://avinetworks.com/glossary/server-load-balancer/)

1. **Task Scheduling Based on LB:** This dynamic algorithm uses a two-level scheduling system and is based on load balancing. It uses the available resources effectively, providing a high level of efficiency. This algorithm distributes the major jobs across the virtual machines, then distributes all of the computer-generated technologies over the host resources, balancing the load and enhancing task retort time, possessions usage, and cloud work out setting efficiency. This process guarantees that the requirements of dynamic users will be met while also maintaining a high ratio of resource usage.

2. **Opportunistic Load Balancing (OLB):** is a fixed procedure that tries to occupy individual node. Because of this, it does cogitate the present job on each node nor does it consider whether or not it is suitable for the task. In other words, independent of the load on the present nodes,

OLB transfers uncompleted jobs to presently vacant nodes in an arbitrary order. The simplicity is advantageous for load balancing, but the lack of consideration for the projected execution times for individual tasks leads to a greater average end period (total cycle time)[71].

3. **Round Robin:** is an itemization of the rounded cycle, in which the initial task is moved to one node, the second to another, and so on until it reaches the last node, at which point it all begins again. The workload of the cluster's nodes is not at all taken into consideration as all tasks in this technique are distributed evenly across all processors, despite the fact that different issues have varying execution times[72]. The approach recognizes the loop as a file and the static time space in round robin arrangement (assignment administration in systems with a time sharing). Only the tasks listed in the time slot and queue may be completed. The work will depart to the line to wait for the succeeding round if it cannot be finished in one time slot. However, picking the right time period can be challenging. The RR scheduling technique performs similarly to FCFS arrangement as soon as the period allocated is quite high. The Round Robin Scheduling technique is also referred to as the processor allocation procedure when a time window is too tiny.

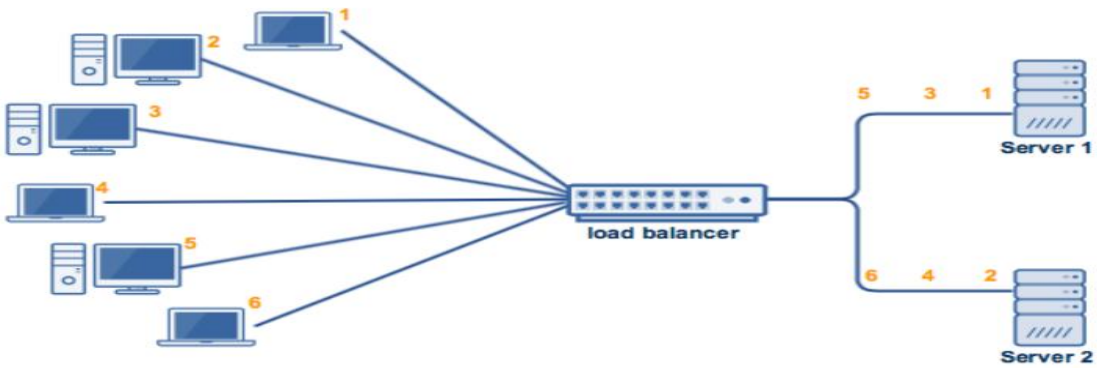

Figure 2.32: Round-Robin Load Balancer

(Source: https:// www.jscape.com/blog/load-balancing-algorithms)

4. **Weighted Round Robin:** is an enhanced refinement of the Round Robin algorithm, where each node is allotted a weight based on performance and capacity. Because nodes with more weight handle more requests, the load is distributed more flexibly as a result.

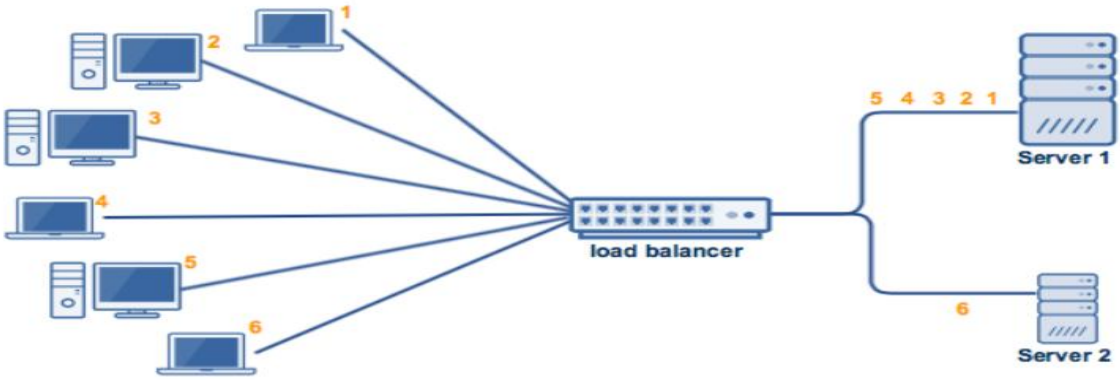

Figure 2.33: Weighted Round-Robin Load Balancer

(Source: https:// www.jscape.com/blog/load-balancing-algorithms)

5. **Randomized** is a static algorithm that chooses one available node at random and sends the current task to it. The algorithm uses a random number generator to choose the node. When every process has the same load, this approach performs well, but there are issues when the load has a varied computational complexity[73]. The deterministic technique is not supported by this algorithm.

6. **Min-Min Algorithm:** is an algorithm for fixed load balancing, which means that the relevant factors linking to the job have been determined in advance. Resources are made available for activities that can be completed as quickly as possible by the algorithm as soon as possible. Each

task's shortest possible execution time is found. The task with the lowest time value is submitted for execution after being found among tasks with the lowest execution times. A queued job will be updated, finished, and performed tasks removed from the queue until all jobs have remained allotted for completion. Tasks with the longest wait times should be given an arbitrary amount of time. The most important drawback of using this technique is the possibility that it will trigger a severe lack of resources. It functions best when the majority of tasks take little time to complete[74].

7. **Max-Min Algorithm:** This algorithm functions almost identically to the min-min algorithm. The key distinction is that this algorithm determines the minimum time required to complete activities before choosing the maximum value, which represents the total time required to complete all jobs across all resources. A further job with a determined maximum execution period is apportioned for use only on the chosen node. The completion times of all jobs on that node are then added together to determine how long they will take to complete. After that, the system deletes the assigned task.

8. **Honeybee Foraging Behavior** is a decentralized technique that uses a local actions server to boost throughput and achieve global load balancing. Calculate the actual VM load, and then determine if the VM is overloaded, under-loaded, or balanced. VMs are organized into groups based on the present load. After the work has been removed from the overloaded VM, the precedence of the job waiting there is taken into concern. The work is then assigned to an under-loaded VM. Earlier completed tasks are helpful for locating under loaded VM. In the following stage, these issues are referred to as bee intelligence agents. The approach shortens task waiting times and VM response times. By increasing system heterogeneity, performance is improved. The fundamental issue is that increasing system size does not result in an increase in bandwidth. When the services of a diverse range of species are required, algorithms are the most suitable.

9. **Active Clustering:** In this method, similar constituents of the structure are assembled together, and those groups work together to complete the process. It operates in the same manner as the method of self-assembled load balancing, which involves rewiring the system with the purpose of accommodating the task pairing system. By connecting these services, a comparable set of assignments is used to improve the system. Resources are improved, which improves system performance. By effectively employing all the resources, bandwidth is increased.

10. **Compare and Balance:** is used to achieve a stability state and manage a load balanced system. In this approach, the present host arbitrarily chooses the lead and likens their loads centered on the likelihood (the quantity of virtual devices running on the present host, as well as the entire cloud system). It sends an additional load to this specific node if the current host's load exceeds that of the chosen host. The identical process is then carried out by each system node. This load balancing technique was created and put into use to speed up VM migration. Shared memory is employed to speed up VM migration.

11. **Lock-free Multiprocessing Solution for LB:** It offered unblocking multiprocessor load balancing, as opposed to existing multiprocessor task management way out, which employ collective memory and fastening to preserve the user's period. The kernel is altered to achieve this. By executing numerous load harmonizing procedures in a single load manager, this method assists to increase the absolute routine of load pairing in multicore systems.

12. **Ant Colony Optimization:** is a distribution algorithm. With each move of the ants, this program dynamically updates resource information. To enable a node to distribute colored colonies over the network, various ant colonies are defined. In a system in which individual ant operates as a mobile representative, carrying an update load harmonizing data to the succeeding

node, daubed ant collections are employed to prohibit movement of ants from similar slot succeeding by one path and to assure their dispersal over all nodes.

13. **Shortest Response Time First:** This algorithm's concept is a direct forwarding. Each process is given a priority in order to be run. Planned FIFO order in processes with equal priorities. The general priority scheduling method has a particular case known as the SRTF algorithm. In the SRTF algorithm, the priority is the opposite of the subsequent CPU burst. As a result, the priority will decrease as the burst processor increases. The task requiring the least processing time is chosen under SRTF policy. Short tasks are completed before long ones in this method. The primary issue with SRTF is that it is crucial to understand or estimate the processing times for each job.

14. **Based Random Sampling**: is a method for scalable and distributed load balancing that achieves self-organization by using a random sample of the system domain. As a result, the load is dispersed throughout all of the scheme's nodes. By adding up to the quantity and similarity of method resources, throughput can be raised by effectively utilizing a wider range of system assets. The process, however, becomes inferior as the multiplicity of possessions increases[75].

15. **The Two Phase Scheduling Load Balancing Algorithm:** The blending of OLB (Opportunistic Load Balancing) and LBMM (Load Balance Min-Min) arranging techniques uses an extraordinary routine operation and system backed up load assessment. To provide load complementary, OLB conserves the operational state of each node. The LBMM scheduling technique is used to diminish the entire end time by reducing the time that each job on the node takes to complete. This method increases efficiency and the effectiveness of resource consumption.

16. **Active Clustering Load Balancing Algorithm:** The algorithm connects related services through local rewiring to optimize tasks. Work by assembling related nodes. Referee nodes are the foundation of the grouping process. Comparable to an initiating node, a referee node specifies connections between neighbors. After that, a referee node cuts off communication with a primary node. The following set of procedures is repeated repeatedly. High resource availability improves system performance, and as a result, bandwidth rises.

17. **ACCLB:** is load assessment approach centered on ant association and the idea of composite network (ACCLB) in open cloud work out. In order to achieve a more even distribution of load, it utilizes low-level conditions and a less sophisticated system. This method helps to increase system performance because it overcomes the lack of homogeneity, is adaptable to a dynamic setting, has strong fault leniency, and has worthy scalability.

18. **Decentralized Content Aware:** is a load-balancing technique also known as the workload and consumer notification procedure. The USP factor is utilized by this technique to direct the exclusive and distinctive qualities of demands besides calculating nodes. The USP aids the manager in selecting the best node for job handling. This method is employed on a distributed source with cheap expenses. By consuming the content details to focus the examination, the system's overall search routine is enhanced, and the idle time of the compute nodes is reduced, increasing their operation.

19. **Server-based LB for Internet-based Spread Services:** is a method of load balancing for web services that are disseminated globally. By implementing a procedure that restricts the sending of demands to the nearest distant servers devoid of congestion, it helps to shorten the

service's response time. For this protocol's implementation commonly uses middleware. Heuristics are also used to aid web servers handle excess load.

20. **Join-Idle-Queue:** is a load balancing method that offers extensive load balancing at distributed senders for dynamically scalable web services. With the aim to shorten the average span of the file for individual processor, it first determines the accessibility of idle processors in individual sender. The approach effectively decreases the system load when it removes the load harmonizing task from the acute route demand procedure. It does not take up any communication weight on the freshly attained job and does not lengthen the real reply period.

21. **Token Routing:** has the sole purpose to diminish the rate of the method by stirring the markers inside the system. However, because of communication bottlenecks in an accessible cloud system, proxies are unable to have ample data to distribute the job. Thus, the distribution of workload among agents is not fixed. By means of empirical approach load assessment centered marker, the procedure's drawback can be eliminated[76]. The routing problem is quickly and efficiently solved by this approach. Agents do not need to be familiar with the universal position or workload neighbors in order to use this approach. Marker transfer agents actually create their own knowledge base so as to make decisions. This knowledge base is derived from tokens that have already been received. Therefore, there are no communications overheads with this method.

22. **Central Queuing:** utilizes the dynamic allocation theory. Every new job that is acknowledged by the file controller is queued. The foremost task is taken out of the line and sent to the asker as soon as a line management receives a request to complete a task. The demand is buffered up until the fresh job is no longer vacant if a queue does not contain any ready tasks.

However, when adding a new job to a queue while there are still open questions, the initial such demand is taken out of the queue and a fresh job is added in its place[77]. The local boot manager notifies the central download manager to start a new task when the CPU consumption drops below the threshold value[77]. If a completed task is found, the manager answers to the request; if not, the order demand is followed prior to the fresh task.

23. **Connection Mechanism:** The weight harmonizing procedure may also be centered on the dynamic scheduling algorithm's minimum amount connections mechanism. In order to compute the quantity of links for respective server in the dynamic task valuation, it is necessary. The amount of connections per server is recorded by the load balancer. When a fresh link is propelled to the server, the number of ports increases, and when the connection is broken or terminated, it drops.

24. **Least Connections:** algorithm uses the server that is currently being serviced by the fewest connections to forward requests to that server. When a server has a minimal quantity of influences, the task stabilizer will make the following request to that server.

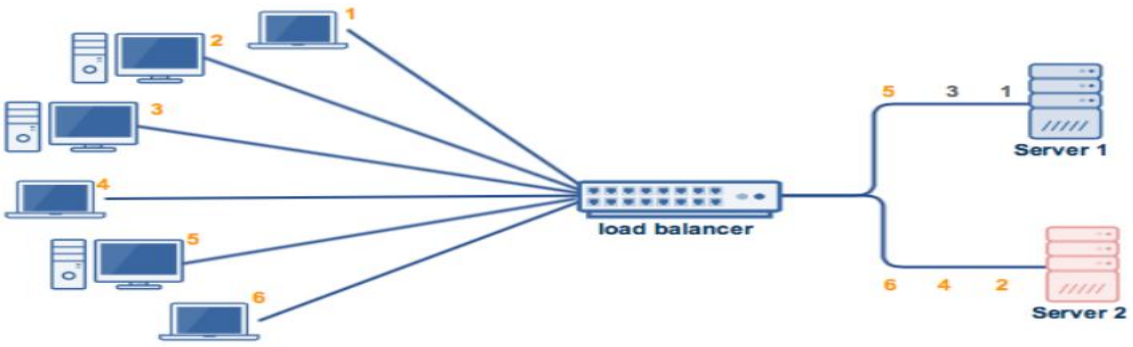

Figure 2.34: Least Connections Load Balancer

(Source: https:// www.jscape.com/blog/load-balancing-algorithms)

25. **Resource Based (Adaptive)**: is a load assessment procedure that demands the setting up of an agent on the request server that communicates the server's current task to the load stabilizer. The application server's deployed agent keeps track of its resources and availability status. To support with load pairing choices, the load stabilizer queries the result of the agent.

26. Resource Based (SDN Adaptive): is a load harmonizing method that integrates data from Layers two, three, four and seven with response from an SDN Controller to provide further effective movement dispersal conclusions. This enables data about the health of the network infrastructure, the position of the servers, and the position of the solutions running on them, and the level of network congestion to all be considered when making load balancing choices[78].

27. Fixed Weighting: Fixed Weighting is a load assessment procedure where the administrator provides a mass to each application server centered on criteria of their choice to show the application server's capacity to handle movement. The entire amount of request will be sent to the application server with the peak mass. All request will be routed to the application server with the next largest mass if the application server with the highest weight fails.

28. Weighted Response Time: In a load harmonizing contrivance known as weighted retort interval, the application server that will get the subsequent request is chosen based on its response time to the previous one. The application server weights are computed based on how quickly a server responded to a fitness check. The demand will be sent to the application server with the quickest response time.

29. Source IP Hash: The origin and target IP addresses of the user and server are joint in the source IP hash task management method to create a special hash key. To assign the user to a specific server, the key is utilized. As the key can be renewed in the event that the period is

terminated, the user's demand is sent to equivalent server it was consuming earlier. When a client must reconnect to a period that is still lively following a discontinuation, this feature is helpful.

2.5 Interconnection Networks

Traditionally, associated interlacement remained characterized as systems that bond several processors. These, however, have undergone a significant evolution over the past 20 years and are now essential in environments like information hubs and high execution handling clusters[79]. The four main categories of topologies for interconnection networks are shared-bus, direct, indirect, and hybrid networks. One of the most crucial aspects in building an interconnection network is selecting the topology. The traffic distribution in the network is governed by the topology that has been established, the routing algorithm, and the workload of the applications.

2.5.1 Fat-Tree Topology

For constructing medium and large structural range setups, the fat-tree methodology is particularly prevalent. The fat-tree approach has several routes connecting hosts, allowing it to offer more accessible data transfer rate than a lone-path hierarchy with equal quantity of nodes[80]. There are switches on the core, aggregation, and edge layers of the conventional three-layer hierarchical tree. The hosts establish connections with the switches at the edge layer. Fat-tree networks' multipath feature gives opportunities to disperse data flow among various network elements.

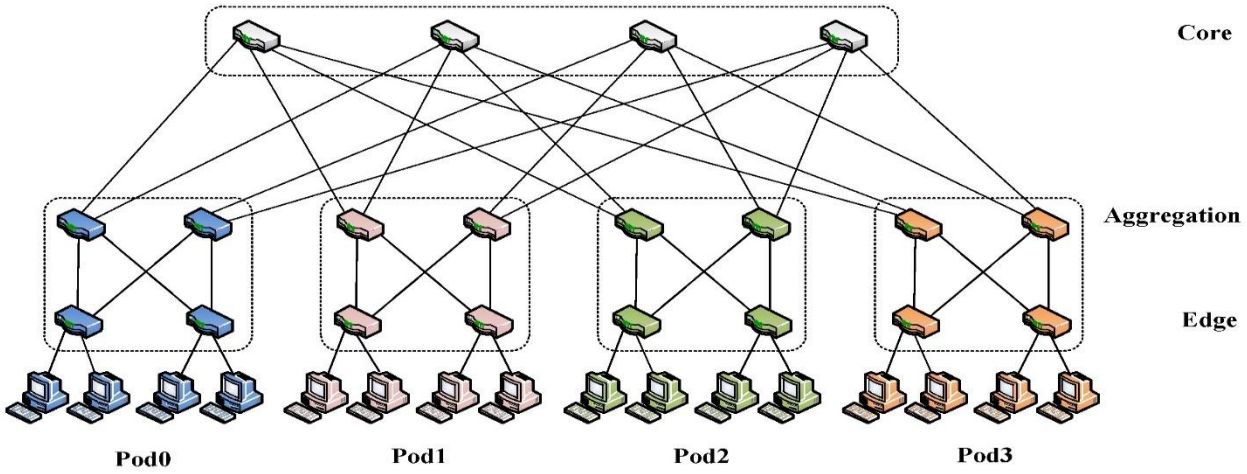

Figure 2.35: Fat-tree network topology

(Source: https:// www.google.com)

The following three characteristics grants fat-trees the preferred approach for peak-enactment communicates:

a) Stalemate independence, using the tree arrangement enables the possibility of routing fat-trees devoid of utilizing simulated straits for impasse averting.
b) Intrinsic fault-acquiescence, the presence of numerous trails amidst distinct source end point sets makes it easier to deal with linkage errors.
c) Complete divisional bandwidth, the system can withstand top pace message among the two share out of the system.

While the fat-tree approach offers extensive connection, it cannot be said that consuming a fat-tree approach alone will result in excellent system functioning because the channeling contrivance also shows a significant part. Factually, the fat-tree approach has been used with adaptive routing, which dynamically constructs a packet's path centered on the position of the network[81]. Nonetheless, routing in the main system area networking technologies currently in use

is deterministic. To effectively utilize the rich connection that the fat-tree topology offers in a structure range link with deterministic channeling, it's crucial to use an effective load stability steering method.

2.6 Summary of Literature Reviewed

In web server environment, it is vital to develop load balancers for equal distribution of nodes in the data center which leads to better resource utilization and fewer failures in the nodes. There has been an extensive number of published reviews regarding load balancing and the algorithms, not fewer than thirty (30) of review papers in this field distributed in the past five years (2018–2022) has been comparatively analysis. There are still a few limitations in the existing review papers, for example, the recent state of art have been reviewed however, and it is a limited explanation as it does not include a comparative analysis. This is an essential feature when doing reviews as it provides deep analysis of the articles and makes it easy for readers to identify areas for improvement in future research. Most review papers do not include flowcharts and thus it does not provide operational flow of the reviewed algorithms. Many papers lack the explanation and evaluation of the performance metrics used in the reviewed articles which is another contribution of the current review paper. It is found that few authors focused on experimental results, the existing review includes a compilation of the experimental results of the reviewed articles based on the simulation platform. This is done to ease the understanding and reduce the hassle to search for such results. This study is structured to review algorithms based on the underlying common algorithms. This way makes it easier for readers that aim to utilize a specific algorithm such as Round Robin, Genetic Algorithm, and so on. This study categorizes algorithms based on their research gap addressing three main aspects in load balancing: Response Time,

Fault Tolerance, others (such as Makespan, Waiting Time, and so on). Additionally, the study proposed a framework to address one of the aspects above, which is fault tolerance.

Endnotes

[1]. S. Jamali, A. Badirzadeh & M.S. Siapoush, ***"On the Use of the Genetic Programming for Balanced Load Distribution in Software-Defined Networks," Digital Communications and Networks,*** **2019**, pp. 288–296.

[2]. X. Shi, Y. Li, H. Xie, T. Yang, L. Zhang, P. Liu, H. Zhang & Z. Liang, ***"An OpenFlow-Based Load Balancing Strategy in SDN," Computers Materials and Continua,*** **2019**, pp. 385–398.

[3]. K. Ramya, M. Sayeekumar & G. Karthik, ***"Software Defined Networking Based Solution in Load Balancing for Media Transfer in Overlay Network," Journal of Computational and Theoretical Nanoscience,*** **2020**, pp. 43–47.

[4]. S. Prabakaran & R. Ramar, ***"Software Defined Network: Load Balancing Algorithm Design and Analysis," The International Arab Journal of Information Technology,*** **2021**, pp. 312-318.

[5]. O. A. Alssaheli, Z. A. Zainal & N.A. Zakaria, ***"Mininet Network Emulator: A Review," International Journal of Computer Science and Network Security,*** **2019**, pp. 147-155.

[6]. I. Afolabi, J. Prados-Garzon, M. Bagaa, T. Taleb & P. Ameigeiras, ***"Dynamic Resource Provisioning of a Scalable E2E Network Slicing Orchestration System". IEEE Trans. Mob. Comput.*** **2020**, pp 2594–2608.

[7]. R. Boutaba, M. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano & O. A Caicedo-Rendon, ***"Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities."*** **J. Internet Serv. Appl.**, **2018**, pp 1–99.

[8]. B. Xiang, J. Elias, F. Martignon & E. Di Nitto, *"Joint Network Slicing and Mobile Edge Computing in 5G Networks". In Proceedings of the ICC 2019-2019 IEEE International Conference on Communications (ICC), Shanghai, China*, **2019**, pp. 1–7.

[9]. A. Ajibare & O. Falowo, *"Resource Allocation and Admission Control Strategy for 5G Networks Using Slices and Users Priorities". In Proceedings of the IEEE AFRICON, Accra, Ghana,* **2019**, pp. 1–6.

[11]. X. Yang, Y. Wang, I. Wong, Y. Liu & L. Cuthbert, *"Genetic Algorithm in Resource Allocation of RAN Slicing with QoS Isolation and Fairness". In Proceedings of the 2020 IEEE Latin-American Conference on Communications (LATINCOM), Santo Domingo, Dominican Republic*, **2020**, pp. 1–6.

[12]. A. Khan, M. Abolhasan, W. Ni, J. Lipman & A. Jamalipour, ***"An End-to-End (E2E) Network Slicing Framework for 5G Vehicular Ad-Hoc Networks". IEEE Trans. Veh. Technol.***, **2021**, pp 7103–7112.

[13]. D.M. Casas-Velasco, O.M.C. Rendon & N. da Fonseca, ***"Intelligent Routing Based on Reinforcement Learning for SoftwareDefined Networking". IEEE Trans. Netw. Serv. Manag.*** **2020**, pp 870–881.

[14]. C. Liang, R. Kawashima & H. Matsuo, ***"Scalable and Crash-Tolerant Load Balancing Based on Switch Migration for Multiple Open Flow Controllers," Second International Symposium on Computing and Networking (CANDAR)***, **2014**, pp 171–177.

[15]. H. Yao, C. Qiu, C. Zhao & L. Shi, ***"A Multicontroller Load Balancing Approach in Software-Defined Wireless Networks," International Journal of Distributed Sensor Networks***, **2015**, pp 1-8.

[16]. L. Shih-Chun, P. Wang & M. Luo, ***"Control Traffic Balancing in Software Defined Networks", Published in: Computer Networks***, **2016**, pp 260- 271.

[17]. A. Neghabi, N. Navimipur, M. Hosseinzadeh & A. Rezaee, ***"Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature," IEEE Access***, **2022**.

[18]. A. Dixit, H. Fang, S. Mukherjee, T.V. Lakshman & R.R. Kompella, *"**ElastiCon: An Elastic Distributed SDN Controller," Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ACM***, **2014**, pp 17-28.

[19]. S. Kaur & J. Singh, ***"Implementation of Server Load Balancing in Software Defined Networking", In Information Systems Design and Intelligent Applications, Springer, New Delhi***, **2016**, pp 147-157.

[20]. R.S. Sutton & A.G. Barto, ***"Reinforcement Learning: An Introduction," 2nd ed.; MIT Press: London, UK***, **2018**.

[21]. H. Li, T. Wei, A. Ren, Q. Zhu & Y. Wang, *"Deep Reinforcement Learning: Framework, Applications, and Embedded Implementations". In Proceedings of the 2017 IEEE/ACM International Conference on Computer-Aided Design, Irvine, CA, USA,* **2017**, pp 847–854.

[22]. D.M. Casas-Velasco, O.M.C. Rendon, & da Fonseca, ***"N.L.S. DRSIR: A Deep Reinforcement Learning Approach for Routing in Software-Defined Networking". IEEE Trans. Netw. Serv. Manag.*** **2021**, pp 1–14.

[23]. A. Sherstinsky, ***"Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network". Phys. D Nonlinear Phenom.*** **2020**.

[24]. K. Arulkumaran, M.P. Deisenroth, M. Brundage & A.A. Bharath, ***"Deep Reinforcement Learning: A Brief Survey". IEEE Signal Process. Mag.*** **2017**, pp 26–38.

[25]. R. Li, Z. Zhao, Q. Sun, C. Yang, X. Chen, M. Zhao & H. Zhang, ***"Deep Reinforcement Learning for Resource Management in Network Slicing". IEEE Access***, **2018**, pp 74429–74441.

[26]. I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini & H. Flinck, ***"Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions". IEEE Commun. Surv. Tutor.*** **2018**, pp 2429–2453.

[27]. Q. Liu, T. Han & E. Moges, *"EdgeSlice: Slicing Wireless Edge Computing Network with Decentralized Deep Reinforcement Learning". In Proceedings of the 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), Singapore*, **2020**; pp 234–244.

[28]. M. Yan, G. Feng, J. Zhou, Y. Sun & Y.C. Liang, ***"Intelligent Resource Scheduling for 5G Radio Access Network Slicing". IEEE Trans. Veh. Technol.*** **2019**, pp 7691–7703.

[29]. G. Liu, Y. Huang, F. Wang, J. Liu & Q. Wang, ***"5G Features From Operation Perspective and Fundamental Performance Validation by Field Trial". China Communication***, **2018**, pp 33–50.

[30]. X. Fu, F.R. Yu, J. Wang, Q. Qi & J. Liao, ***"Service Function Chain Embedding for NFV-Enabled IoT Based on Deep Reinforcement Learning," IEEE Communications Magazine,*** 2019, pp 102-108.

[31]. J. Du, C. Jiang, Z. Han, H. Zhang, S. Mumtaz, & Y. Ren, ***"Contract Mechanism and Performance Analysis for Data Transaction in Mobile Social Networks," IEEE Transactions on Network Science and Engineering,*** **2019**, pp 103-115.

[32]. P. Zhang, H. Yao, M. Li & Y. Liu, ***"Virtual Network Embedding Based on Modified Genetic Algorithm," Peer-to-Peer Networking and Applications***, **2017**, pp 1-12.

[33]. H. Cao, S. Wu, Y. Hu, Y. Liu & L. Yang, ***"A survey of Embedding Algorithm for Virtual Network Embedding," China Communications,*** **2019**, pp 1-33.

[34]. P. Zhang, H. Yao & Y. Liu, ***"Virtual Network Embedding Based on Computing, Network, and Storage Resource Constraints," IEEE Internet of Things Journal,*** **2018**, pp 3298-3304.

[35]. J. Du, C. Jiang, H. Zhang, Y. Ren & M. Guizani, ***" Auction Design and Analysis for SDN-Based Traffic Offloading in Hybrid Satellite-Terrestrial Networks," IEEE Journal on Selected Areas in Communications***, **2018**, pp 2202-2217.

[36]. Y. Liu, B. Zhao, P. Zhao, P. Fan & H. Liu, ***"A Survey: Typical Security Issues of Software-Defined Networking," China Communications,*** **2019**, pp 13-31.

[37]. P. Zhang, C. Wang, Z. Qin & H. Cao, ***"A Multi-Domain Virtual Network Embedding Algorithm Based on Multi-Objective Optimization for Internet of Drones Architecture in Industry 4.0," Softw Pract Exper,*** **2020**, pp 1-19.

[38]. P. Zhang, "***Incorporating Energy and Load Balance into Virtual Network Embedding Process," Computer Communications,*** **2018**, pp 80-88.

[39]. C. Jiang, H. Zhang, Y. Ren, Z. Han, K. Chen, & L. Hanzo, ***"Machine Learning Paradigms for Next-Generation Wireless Networks," IEEE Wireless Communications,*** **2017**, pp 98-105.

[40]. M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, & J. van der Merwe, *"Design and Implementation of a Routing Control Platform". In Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2, Ser. NSDI'05. Berkeley, CA, USA: USENIX Association,* **2005**, pp 15–28.

[41]. J. Biswas, A. A. Lazar, J. F. Huard, K. Lim, S. Mahjoub, L. F. Pau, M. Suzuki, W. Wang, & S. Weinstein, ***"The IEEE P1520 Standards Initiative for Programmable Network Interfaces," Comm. Mag.***, **1998**, pp 64–70.

[42]. B. Schwartz, A. Jackson, W. Strayer, W. Zhou, R. Rockwell, & C. Partridge, *"Smart Packets for Active Networks,". In Open Architectures and Network Programming Proceedings, 1999. OPENARCH'99. 1999 IEEE Second Conference*, **1999**, pp 90–97.

[43]. D. Wetherall, V. J. Guttag, & D. Tennenhouse, ***"Ants: A Toolkit for Building and Dynamically Deploying Network Protocols". In Open Architectures and Network Programming,* 1998**, pp 117–129.

[44]. M. Sune, V. Alvarez, T. Jungel, U. Toseef, & K. Pentikousis, *"An OpenFlow Implementation for Network Processors". In Third European Workshop on Software Defined Networks*, **2014**.

[45]. D. Parniewicz, R. Doriguzzi Corin, L. Ogrodowczyk, M. Rashidi Fard, J. Matias, M. Gerola, V. Fuentes, U. Toseef, A. Zaalouk, B. Belter, E. Jacob, & K. Pentikousis, *"Design and Implementation of an OpenFlow Hardware Abstraction Layer". In Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing, Ser. DCC '14. New York, NY, USA: ACM,* **2014**, pp 71–76.

[46]. B. Belter, D. Parniewicz, L. Ogrodowczyk, A. Binczewski, M. Stroinski, V. Fuentes, J. Matias, M. Huarte, & E. Jacob, *"Hardware Abstraction Layer as an SDN-Enabler for Non-OpenFlow Network Equipment". In Third European Workshop on Software Defined Networks,* **2014**.

[47]. B. Belter, A. Binczewski, K. Dombek, A. Juszczyk, L. Ogrodowczyk, D. Parniewicz, M. Stroinski, & I. Olszewski, *"Programmable Abstraction of Datapath". In Third European Workshop on Software Defined Networks,* **2014**.

[48]. C. Peng, M. Kim, Z. Zhang, & H. Lei, ***"VDN: Virtual Machine Image Distribution Network for Cloud Data Centers". In INFOCOM, 2012 Proceedings IEEE,* 2012**, pp 181–189.

[49]. Z. Zhang, Z. Li, K. Wu, D. Li, H. Li, Y. Peng, & X. Lu, ***"VM Thunder: Fast Provisioning of Large-Scale Virtual Machine Clusters," Parallel and Distributed Systems, IEEE Transactions*, 2014**, pp 1–1.

[50]. R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, & G. Parulkar, ***"FlowVisor: A Network Virtualization Layer," Deutsche Telekom Inc. R&D Lab, Stanford, Nicira Networks, Tech. Rep.*, 2009**.

[51]. A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, & S. Krishnamurthi, *"Participatory Networking: An API for Application Control of SDNs". In Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, ser. SIGCOMM '13. New York, NY, USA: ACM*, **2013**, pp 327–338.

[52]. F. Botelho, A. Bessani, F. Ramos, & P. Ferreira, *"On the Design of Practical Fault-Tolerant SDN Controllers". In Third European Workshop on Software Defined Networks,* **2014.**

[53]. S. Matsumoto, S. Hitz, & A. Perrig, *"Fleet: Defending SDNs from Malicious Administrators". In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, Ser. HotSDN '14. New York, NY, USA: ACM,* **2014**, pp 103–108.

[54]. P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, & G. Gu, *"A Security Enforcement Kernel for OpenFlow Networks". In Proceedings of the First Workshop on Hot Topics in Software Defined Networks, Ser. HotSDN '12. New York, NY, USA: ACM,* **2012**, pp 121–126.

[55]. L. Richardson & S. Ruby, ***"RESTful Web Services". O'Reilly Media, Inc.***, **2008**.

[56]. T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, & S. Shenker, *"Practical Declarative Network Management". In Proceedings of the 1st ACM Workshop on Research on Enterprise Networking, Ser. WREN '09. New York, NY, USA: ACM,* **2009**, pp 1–10.

[57]. M. Alizadeh, T. Edsall, & S. Dharmapurikar, *"CONGA: Distributed Congestion-Aware Load Balancing for Datacenters," ACM Conference on SIGCOMM. ACM,* **2014**, pp 503-514.

[58]. S. Attarha, K. Haji Hosseiny, G. Mirjalily & K. Mizanian, *"A Load Balanced Congestion Aware Routing Mechanism for Software Defined Networks," 2017 Iranian Conference on Electrical Engineering (ICEE), Tehran,* **2017**, pp 2206-2210

[59]. C. Wang, B. Hu, S. Chen, D. Li & B. Liu, ***"A Switch Migration-Based Decision-Making Scheme for Balancing Load in SDN". In IEEE Access,*** **2017**, pp 4537-4544

[60]. S. Kaur & J. Singh, ***"Implementation of Server Load Balancing in Software Defined Networking". In Information Systems Design and Intelligent Applications, Springer, New Delhi,*** **2016**, pp 147-157.

[61]. D. Perepelkin & V. Byshov, *"Visual Design Environment of Dynamic Load Balancing in Software Defined Networks," 27th International Conference Radioelektronika (RADIOELEKTRONIKA),* **2017**, pp 1-4.

[62]. A. Basta, A. Blenk, & H. B. Hassine, *"Towards a dynamic SDN virtualization layer: Control path migration protocol," International Conference on Network and Service Management,* **2015**, pp 354-359.

[63]. J. Cui, Q. Lu, H. Zhoang, M. Tian, & L. Liu, ***"A Load Balancing Mechanism for Distributed SDN Control Plane Using Response Time", IEEE Transactions on Network and Service Management,*** **2018**. pp 134-136

[64]. M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, & A. Vahdat***, "Hedera: Dynamic Flow Scheduling for Data Center Networks," NSDI,*** **2010**, pp 19.

[65]. A. Tsuchiya, F. Fraile, I. Koshijima, A. Ortiz, & R. Poler, ***''Software Defined Networking Firewall for Industry 4.0 Manufacturing Systems,'' J. Ind. Eng. Manage.***, **2018**, pp 318–333.

[66]. D. E. Eisenbud***, ''Maglev: A fast and Reliable Software Network Load Balancer''. In Proc. 13th USENIX Symp. Netw. Syst. Design Implement. (NSDI),*** **2016**, pp 523–535.

[67]. A. A. Neghabi, N. J. Navimipour, M. Hosseinzadeh, & A. Rezaee, ***''Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature,'' IEEE Access,*** **2018**, pp 14159–14178.

[68]. A. A. Neghabi, N. J. Navimipour, M. Hosseinzadeh, & A. Rezaee, ***''Nature-Inspired Meta-Heuristic Algorithms for Solving the Load Balancing Problem in the Software-Defined Network,'' Int. J. Commun. Syst.***, **2019**.

[69]. P. T. Congdon, P.v, M. Farrens, & V. Akella, ***"Simultaneously Reducing Latency and Power Consumption in OpenFlow Switches", IEEE/ACM Trans. Netw.***, **2014**, pp 1007–1020.

[70]. B. Pfaff, ***"The Design and Implementation of Open Vswitch". In Proc. 12th USENIX Symp. Netw. Syst. Design Implement. (NSDI)***, **2015**, pp 117–130.

[71]. V. B. Harkal & A. Deshmukh, ***"Software Defined Networking with Floodlight Controller," Int. J. Comput. Appl.***, **2016**, pp 8887.

[72]. D. Erickson, ***"The Beacon Openflow Controller". In Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)***, 2013, pp. 13–18.

[73]. T. Guesmi, A. Kalghoum, B.M. Alshammari, H. Alsaif & A. Alzamil, ***"Leveraging Software-Defined Networking Approach for Future Information-Centric Networking Enhancement," Symmetry,*** **2021**, pp 1-19.

[74]. G. Singh & K. Kaur, ***"An Improved Weighted Least Connection Scheduling Algorithm for Load Balancing in Web Cluster Systems," International. Research. Journal. Engineering. Technology***, **2018**, pp 1950–1955.

[75]. C.-T. Yang, S.-T. Chen, J.-C. Liu, Y.-W. Su, D. Puthal, & R. Ranjan, ***"A Predictive Load Balancing Technique for Software Defined Networked Cloud Services," Computing***, **2019**, pp 211–235.

[76]. T. Semong, T. Maupong, S. Anokye, K. Kehulakae, S. Dimakatso, G. Boipelo & S. Sarefo, ***"Intelligent Load Balancing Techniques in Software Defined Networks: A Survey," Electronics***, **2020**, pp 1-24.

[77]. D. Shen, W. Yan, Y. Peng, Y. Fu & Q. Deng, ***"Congestion Control and Traffic Scheduling for Collaborative Crowdsourcing in SDN Enabled Mobile Wireless Networks," Wireless Communications and Mobile Computing***, **2018**, pp 1-12.

[78]. Q. Youssef, M. Yassine & A. Haqiq, ***"Secure Software Defined Networks Controller Storage Using Intel Software Guard Extensions," International Journal of Advanced Computer Science and Applications***, **2020**, pp 475-481.

[79]. H. Polat & O. Polat, ***"An Intelligent Software Defined Networking Controller Component to Detect and Mitigate Denial of Service Attacks". Journal of Information and Communication Technology***, **2021**, pp 57-81.

[80]. H. Babbar, S. Rani, D. Gupta, H.M. Aljahdali, A. Singh & F. AlTurjman, ***"Load Balancing Algorithm on the Immense Scale of Internet of Things in SDN for Smart Cities," Sustainability***, **2021**, pp 1-16.

[81]. N. Fotiou, ***"Information-Centric Networking (ICN)," Future Internet***, **2020**, pp 1-2.

Chapter 3

Methodology

3.1 Research Approach

This section explains the proposed framework for improvised load balancing in web server environment. Furthermore the procedure implemented to realize a dynamic load balancing method, and as well other constituents and software implements utilized in this study to establish the testbed The main goal of the proposed framework is to provide a high availability web server environment whereby it avoids system failures and recovering of user tasks which in turn enhances the security in the web server environment. This framework resolves fault tolerance issues, increases throughput and reduces response time in the server by using dual load balancers in the hosting environment and applying migration technique which has not been addressed in the previous literature. The user request is analyzed and passed to the selected data center based on the availability of resources. Those servers (VMs) should not be overloaded or under-loaded, there is a need for an equal distribution among them. This is where the load balancing concept comes into place. To keep up the performance of the web server an efficient load balancing algorithm must be utilized. Unequal load distribution could exist through many factors one of them is wrong task scheduling. Without proper task scheduling technique, the resources will not be efficiently utilized.

As illustrated in figure 3.1 below, the proposed framework consists of two layers:

Top Layer: deals with requests from multiple different clients (application's users) of both mobile and desktop. Users can access the internet using multiple different devices to send requests to the server. In the web server environment, data center (DC) can be described as a big storage for web servers and data. DC receives requests and sends them to the active load balancer. In the top layer of the framework, there are two types of load balancers: Active Load

Balancer and Passive Load Balancer. The purpose of providing a passive load balancer is in case of failure in the main/active load balancer. Failures could exist for many reasons such as overloading the VMs with many requests or unnecessary migration of requests.

Bottom Layer: deals with allocation of user requests to VMs. As can be seen from figure 3.1, in the primary VMs batch, VM3's status is overloaded. Thus, a migration technique should be applied to transfer the failed requests to another available VM found in the secondary VMs batch. In a situation of having a system failure in the active load balancer, this will then turn off the active load balancer and declares it as unavailable which can cause severe downtime of the system in the cloud. However, in this case, the passive load balancers can take over and continue to re-allocate requests to available VMs. The allocation table is then updated whenever a VM becomes available, overloaded, idle and the number of requests it has been allocated.

The proposed framework makes use of replication concepts. Providing a standby load balancer can address fault tolerance and availability issues in the web server environment which has been identified as a research gap in this study. A passive Load Balancer must be available to take over when there is a failure therefore it must be configured similarly to the active load balancer. VMs should have the same workload to function properly and in case of an overloading situation, a migration technique must be applied to transfer requests.

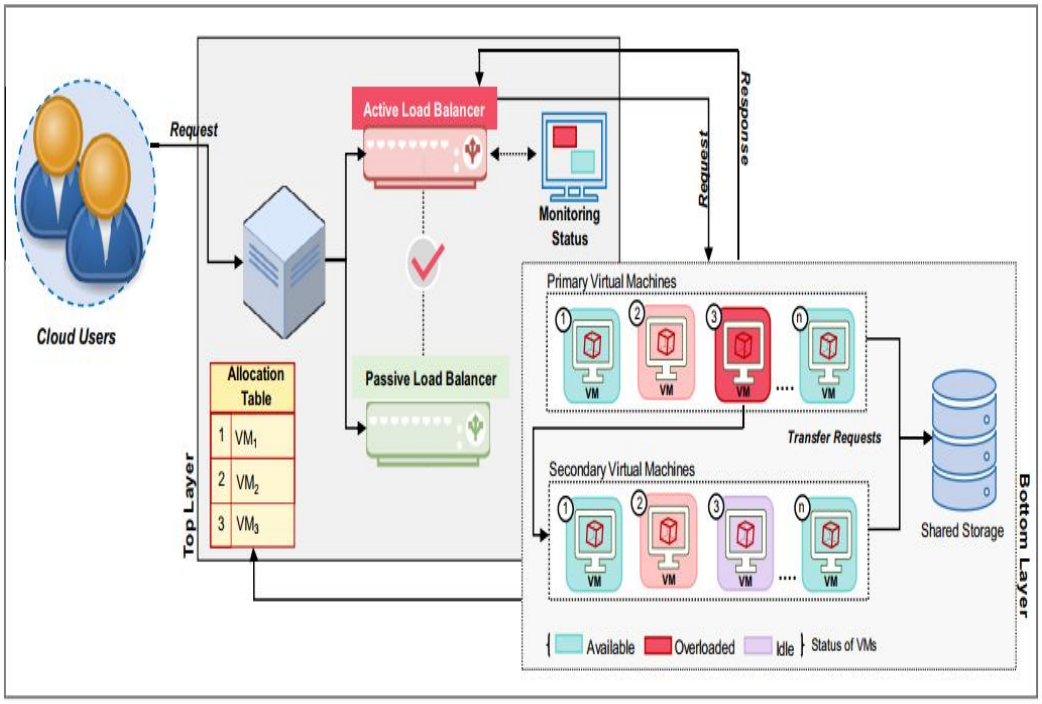

Figure 3.1: Proposed Framework with Fault-Tolerant Capability

(Source: Research Design 2022)

3.2. System Design

In order to exploit the resource competence of each connection in a network, the load balancing algorithm's task is to balance traffic from incoming and outgoing network flows. Maintaining awareness of the network's current condition is important to accomplish this goal. The phases of the load balancing algorithm are illustrated in Figure 3.2 below.

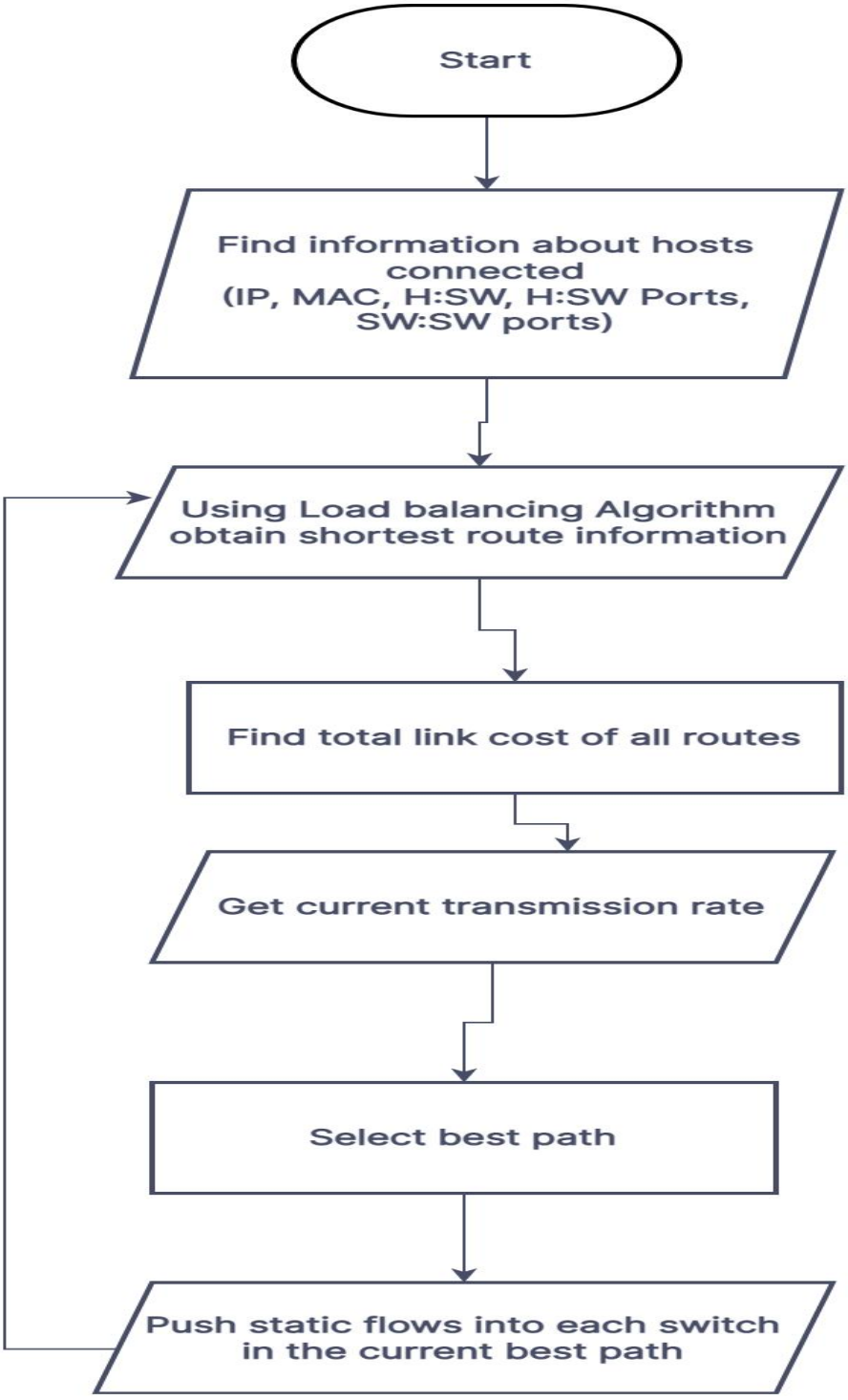

Figure 3.2: Dynamic Load Balancing Algorithm

(Source: Research Design 2022)

The algorithm's initial phase is to gather operational data on the network structure and the gadgets therein which includes IP addresses, MAC addresses, ports, connections, etc. The succeeding phase is to locate path data based on the load-balancing algorithm. Here, the search needs to be limited to a lesser area of the Fat-Tree network interlinks in order to identify the shortest routes between the basis and target hosts. Next, calculate the overall connection cost for each of these routes between the root and target hosts. After calculating the diffusion overheads of the links, the flows are formed based on the least diffusion rate of the links at the time. The best route is chosen based on the cost, and inert movements are then advanced into respective switch in the chosen fitting route. As a result, each switch along the chosen path will have the required flow entries to execute the exchange amidst the two termination spots. To end with, the application keeps updating this data every minute, causing it to be dynamic.

3.3 Implementation Overview

A test-bed has been set up in this study in Linux, utilizing Mininet software to simulate the network, the open-source OpenDaylight platform (ODL) as the SDN controller, Python language to design the fat-tree topology and create the load balancing set of rules program, and iPerf to measure network functioning. The steps in design are shown in the diagram below.

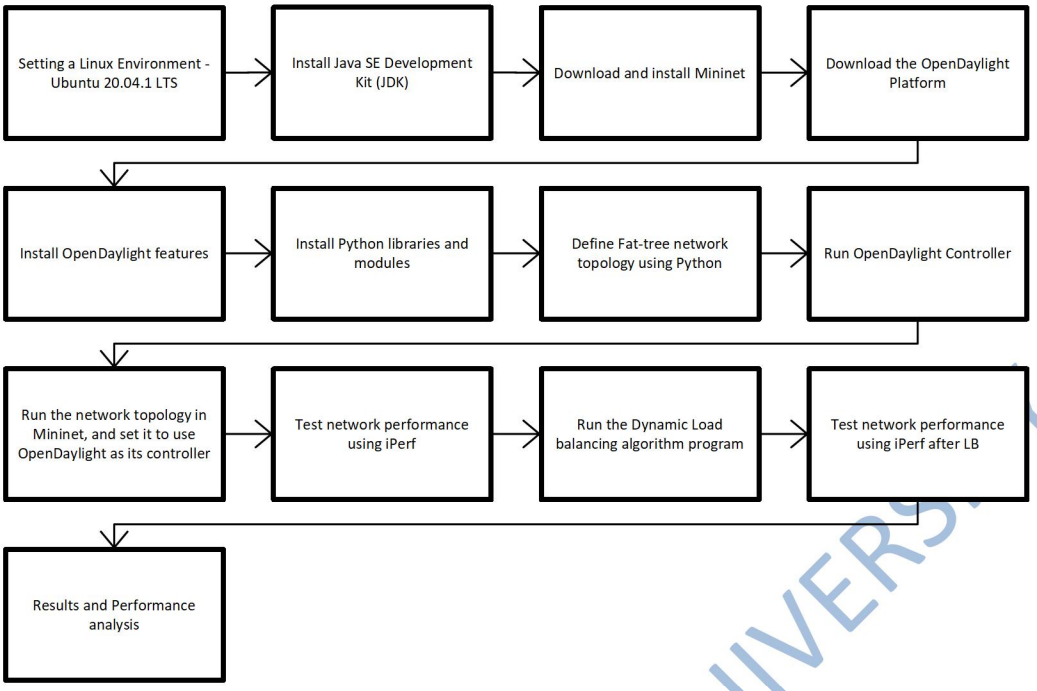

Figure 3.3 Implementation Overview

(Source: Research Design 2022)

3.4 Requirement Specification

3.4.1 Mininet

Mininet is a network emulator that enables rapid prototyping of huge networks on a single host computer. On a single Linux kernel, it manages a variety of end hosts, switches, routers, and links. By executing the same kernel, system, and user code on a single system, it leverages lightweight virtualization to make the system appear to be a full network.

Mininet main advantages:

1. Mininet is an open source project.
2. Custom topologies can be created.

3. Mininet runs real programs.
4. Packet forwarding can be customized.

In contrast to simulations, Mininet executes actual, unmodified code, including application code, OS kernel code, and control plane code (both OpenFlow controller code and Open vSwitch code), and it connects to real networks with ease.

It is necessary to have a controller machine running on a system-working framework, such as ODL or POX, in order to create the SDN load balancer. With the programming interface in OpenFlow, the crucial components of the forwarder have a separate control arrangement. For SDN, a comprehensive physical foundation that connects every component via a secure channel is required, and this is costly.

This project makes use of a specific test system to handle this problem. Mininet is a tool that simulates Software Defined Networks, which allow for quick prototyping of a sizable virtual foundation system using a desktop PC. In light of programming, it facilitates the use of virtual models of flexible systems. For instance, OpenFlow quickly associates and updates models for Software Defined Networks by using a rudimentary virtualization operating system using these primitives[1]. A few features of the Mininet are the following.

1. It permits that various scientists autonomously test the same system topology.
2. It permits the testing of a complex topology without the need of a physical system.
3. It incorporates apparatuses to troubleshoot and run tests over the system.
4. It underpins various topologies and incorporates an essential arrangement of them.
5. It gives straightforward Python APIs for making and testing systems.

6. It gives a straightforward and modest path for testing systems for the improvement of OpenFlow.

Mininet is more compatible than simulators since it uses the application's original code, as opposed to simulators, which use test code. This facilitates testing and running prior to implementation in production. The OpenFlow procedure is used in the programming of Mininet switches, which makes it effortless to examine and amend the code as needed. By issuing a single command, it constructs a realistic virtual network that functions on an actual kernel, switch, and application code on any virtual machine (VM), cloud, or native platform[2]. According to the OpenFlow website, Mininet is a useful tool for developing, sharing, and experimenting with OpenFlow and Software-Defined Networking systems.

3.4.2 OpenDay Light Controller

The OpenDaylight controller (ODL) is a multi-protocol controller architecture designed for SDN dispositions on contemporary varied multi-vendor links. It is highly accessible, flexible, extendable, scalable, and available in several languages. ODL is a platform for model-driven facility construct that enables users to quickly create applications that run on a variety of south-bound protocols and hardware.

Additionally, it has internal plugins that expand the network's capabilities. For instance, it features dynamic plugins that make it possible to acquire both network topology and statistics.

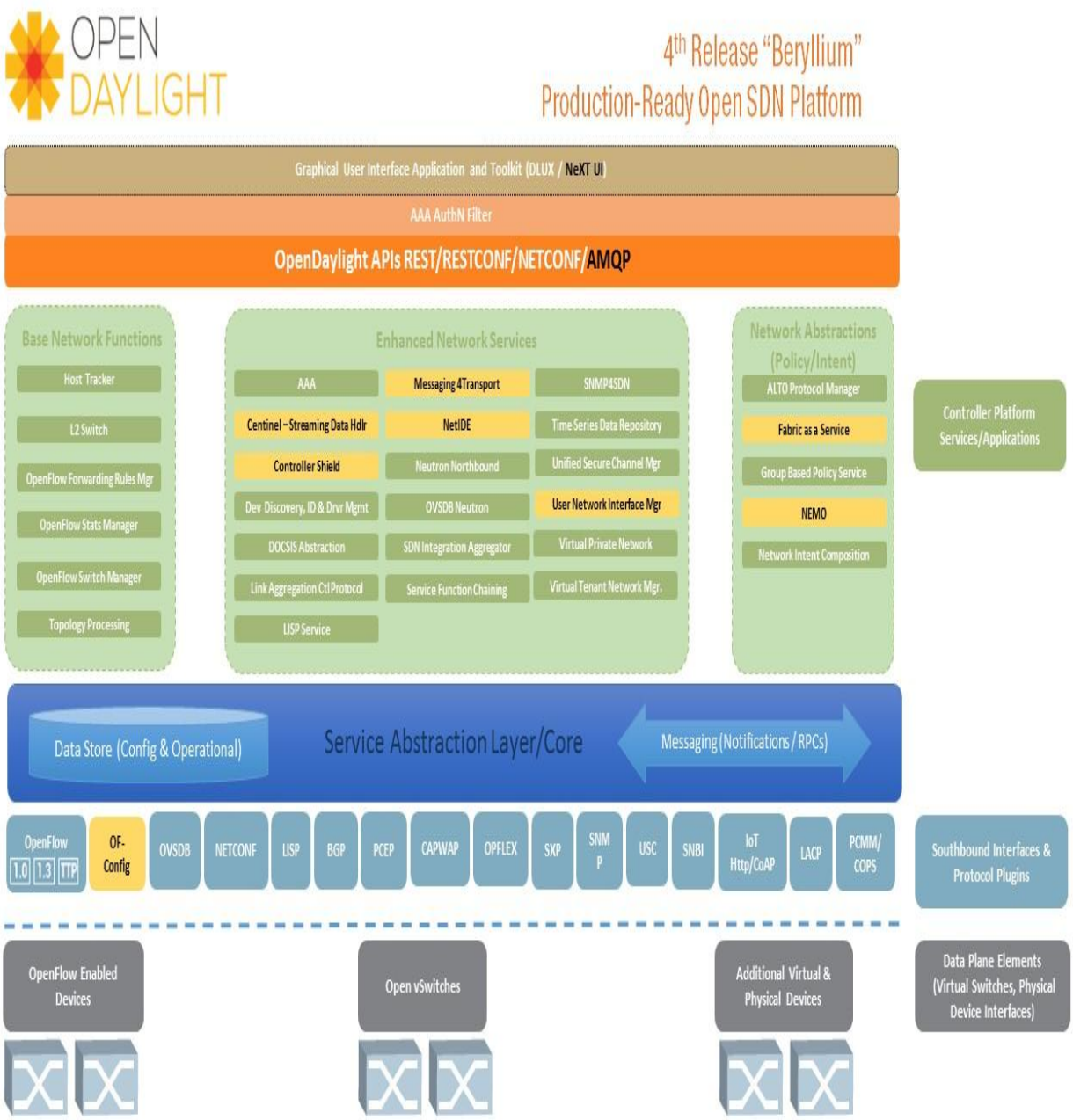

Figure 3.4 Beryllium-SR4 architecture framework

(Source: https://www.opendaylight.org/)

3.4.3 iPerf

iPerf is a well-known network assessing device for evaluating the effectiveness of a network link and the bandwidth operation of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The user is capable of completing a quantity of assessments that give perception into the network's bandwidth disposal, latency, jitter, and data loss by changing various timing, buffer, and protocol parameters (TCP, UDP, SCTP with IPv4 and IPv6)[3]. iPerf is open source software that works on a variety of operating systems, including Linux, UNIX, and Windows.

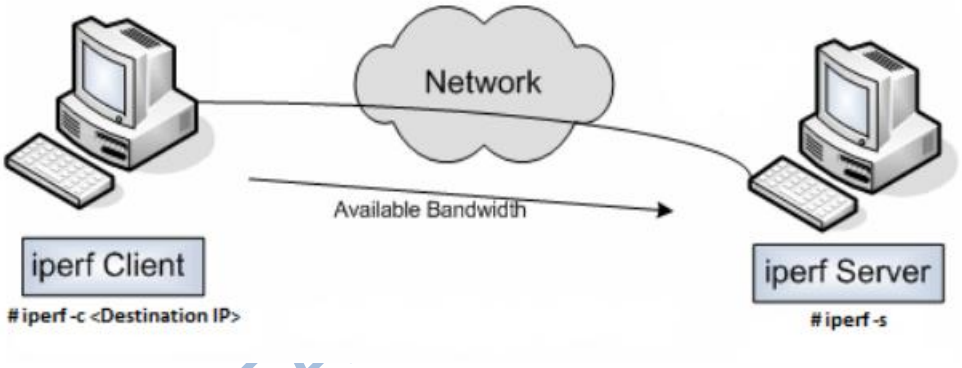

Figure 3.5 iPerf Bandwidth measurement.

(Source: https://www.google.com/)

3.4.4 Programming Language Used: Python

Python was used in this study to express the Fat-tree topology in Mininet and to create the application for the load balancing technique. Python is an object-oriented, interpreted language that may be used for a variety of tasks. It has a flexible dynamic type system, strong high-level data structures, and a clear, understandable syntax[4]. Python is a flexible language that

can be applied interactively, in standalone scripts, for complex systems, or as an add-on to already existing applications. Windows, Macintosh, and Linux computers can all run the language.

Python can be easily extended through C or C++ modules, and it is also capable of been used as a library in other programs. Additional system-specific extensions are available. There is also a sizable library of common Python modules. Python instructs are substantially littler and therefore are quicker to devise than C applications. Python code is simpler to deliver, script, and support than Perl code. Python is more appropriate for copious or more complex projects than TCL is.

3.5 Research Methods

Round-robin and weighted round-robin are the two most popular load balancing algorithms. These algorithms, however, are static, which means that they are not dynamically modified in response to incoming requests. Regardless of the scope of the work or the administering influence of the resources, the RR algorithm distributes the incoming requests in a circular pattern to the available resources. The WRR algorithm takes the processing capability of resources into consideration, and the resource with better processing power is given a larger quantity of incoming jobs. The dynamic WRR algorithm is also used for comparison with these algorithms. This method uses the waiting time to determine the best server for each new request that is made. Using data on the server's processing capacity, task length, task priority, and processing time needed for servers to complete jobs with equal or higher priorities, this algorithm aids in the selection of the most appropriate server. Information regarding a task, including its duration and priority, is recorded once it enters the system. The weight of each

server is then determined, as was previously described. The incoming job is then sent with additional weight to the server. The weight of a server essentially relates to the length of time needed for that particular server to finish the tasks with equal or higher priorities in its queue. If the completion time is quicker, the weight of the server increases, and vice versa. Task migration is carried out when a task's runtime varies significantly from expected values. By determining which server has the least number of jobs with the same or higher priority, the high priority task from the overloaded server's queue is sent to the under-loaded or ideal server[5]. This speeds up the completion of high-priority tasks, and the migration helps to improve the efficient use of resources.

3.5.1 Waiting Time Calculation

According to the waiting time to complete the subsequent incoming job, each VM is given a weight in the proposed method. A VM's weight and waiting time have an inverse relationship; a VM has a high weight when the waiting time is short and vice versa. The proposed method bases the calculation of waiting time on the importance of an incoming job. Each task has a priority value between 1 and 10, and one with a lower integer value is regarded as having a high priority. The following formula is used to determine each VM's waiting time, WT_{vm}:

$$WT_{vm} = \sum_{p=1}^{i} T_p$$

Equation (3.1)

where T is a task's execution time and p is the task priority. When a VM gets an incoming task with priority I the waiting time is determined using the equation above. In this experiment, the cloudlet with the lowest priority is regarded as having a high priority, while the cloudlet with the

highest priority is regarded as having a low priority[6]. For instance, if a job with priority 5 is received, the waiting times for tasks with priorities 1, 2, 3, 4, and 5 are determined. It determines the sum of all tasks' execution times whose priority is equal to or higher than the entering tasks.

The execution time of a task is calculated as follows:

$$T = \frac{S}{V_m}$$

Equation (3.2)

where S represents the cloudlet size in MI (Million Instructions) and V_m represents the processing capacity of a VM in MIPS (Million Instructions Per Second).

3.5.2 Resource Load Calculation

To conduct task migration between virtual machines, the load of all VMs must be calculated, and it must be split into three categories: overloaded, under-loaded, or balanced. Calculating the overall processing capacity of all VMs (C), threshold variance for each VM, and threshold for each VM are all necessary steps in doing this categorization[7]. The threshold range is computed using two variances, V_{min} and V_{max}. These numbers represent a machine's percentage usage; in this experiment, we utilized a minimum value of 0.7 and a maximum value of 0.9.Total capacity of all virtual machines is calculated as follows:

$$C = \sum_{i=1}^{k} c_i$$

Equation (3.3)

where k represents the number of available virtual machines and c represents the processing capacity of a VM. Threshold for each VM is calculated as follows:

$$T_{min} = \frac{c}{C} * V_{min} * L * n$$

Equation (3.4)

where V_{min} represents the minimum variance, L represents the total capacity of a node, and n represents the total number of virtual machines. T_{min} represents the minimum threshold value of a VM.

$$T_{max} = \frac{c}{C} * V_{max} * L * n$$

Equation (3.5)

where V_{max} represents the minimum variance and T_{max} represents the maximum threshold value of a VM.

The VMs are classified by using the following equations:

$$WT_{vm} < T_{min},\ Underloaded$$

$$WT_{vm} > T_{max},\ Overloaded$$

$$WT_{vm} = [T_{min}, T_{max}],\ Balanced$$

Equation (3.6)

Any virtual machine (VM) that exceeds the threshold level is regarded as being overloaded. A VM with a load that is below the threshold level receives the task migration from an overloaded VM. The under-loaded VM is prepared to accept the task until it hits the threshold level. No

migration is conducted if there are no under-loaded VMs. According to the proposed method, the VM that has the least number of tasks with equal or higher priorities is chosen as the under-loaded VM.

3.5.3 Implementation of Round-Robin Algorithm

The round-robin algorithm's step-by-step procedure is given in Algorithm 1. Among static algorithms, this is one of the most frequently applied. The incoming requests are cycled through the servers according to this algorithm. The first request is assigned to any random server, and the following requests are processed in cyclic order. For perfect static conditions, cloud service providers frequently utilize this algorithm.

Algorithm 1 Round-Robin Algorithm Pseudocode

1. The first incoming request is moved by the scheduler to the ready queue.
2. The data center controller chooses which server will receive the first request.
3. Any server, chosen at random, receives the first request.
4. Servers are arranged cyclically once the first request has been assigned.
5. The server that received the first request gets moved back to all servers.
6. The subsequent request is passed on to the following server in a circular direction.
7. Step 3 is repeated for every request until the scheduler completes processing them all.

3.5.4 Implementation of Weighted Round-Robin Algorithm

The weighted round-robin algorithm's step-by-step procedure is given in Algorithm 2. The RR method has been improved by this algorithm. When scheduling a task in RR, the processor or

server's processing power is not taken into account. However, the WRR algorithm determines each server's weight based on its processing capacity. Servers that have more processing power weigh more, while servers that have less processing power weigh less. Servers are ranked according to weight when the weight is determined, and those with higher weight are given more tasks than the other servers.

Algorithm 2 Weighted Round-Robin Algorithm Pseudocode

1. The first incoming request is transferred to the ready queue by the scheduler.
2. Each server is assigned a weight based on its processing power; a server with more processing power is assigned a heavier weight.
3. In a cyclical arrangement, servers are grouped in decreasing weight order from high to low.
4. Designate a server with more weight to receive an incoming job or cloudlet. A greater number of tasks are given to servers with more weight than to the others.
5. After the server receives the desired requests, it switches back to other servers.
6. The subsequent request is cycled to the next server.
7. Repeat step 4 until the scheduler completes processing every request.

3.5.5 Implementation of Dynamic Weighted Round-Robin Algorithm

The dynamic weighted round-robin algorithm's detailed process is listed in Algorithm 3. The WRR algorithm has been enhanced by this algorithm. Once the weight is assigned to the servers in the WRR algorithm, incoming jobs are assigned in the same order. The load on each server at the time is not taken into account. However, the dynamic weighted round-robin algorithm

calculates each server's waiting time for each request, and then assigns incoming requests to those servers that have a shorter wait time.

Algorithm 3 Dynamic Weighted Round-Robin Algorithm Pseudocode

1. Determine the duration of an incoming task or request.
2. Based on the available loads on each server, determine the waiting times for each server.
3. Each server is given a weight based on the waiting time; a server with a shorter wait time will be given a higher weight.
4. Assign incoming tasks to servers that have shorter wait times.
5. Calculate the threshold and load for each server when the incoming task has been assigned.
6. Should a server be discovered to be overloaded:
    a. Pick any task from the task waiting queue of the server.
    b. Identity the under loaded servers.
    c. Select the server which has less waiting time.
    d. Continue the process till it has overloaded servers.
7. Go to step 2 for each request until the scheduler processes all requests.

3.5.6 Algorithm: Round Robin Algorithm for Load Balancing

Table 3.1: Round Robin Algorithm

RoundRobin.sched_Proc(self, proc_data, t_s)

```
def sched_Proc(self, proc_data, t_s):
    time_St = []
    time_exited = []
```

```
    exec_proc = []
    rd_que = []
    s_time = 0
    proc_data.sort(key=lambda x: x[1])
    '''
    process Sorting by Arrival Time
    '''
    while 1:
        nor_que = []
        temp = []
        for i in range(len(proc_data)):
            if proc_data[i][1] <= s_time and proc_data[i][3] == 0:
                present = 0
                if len(rd_que) != 0:
                    for k in range(len(rd_que)):
                        if proc_data[i][0] == rd_que[k][0]:
                            present = 1
                '''
                 Verify that the subsequent procedure is not part of rd_que
                '''
                if present == 0:
                    tem.ext([proc_data[i][0], proc_data[i][1], proc_data[i][2], proc_data
[i][4]])
                    rd_que.append(temp)
                    temp = []
                '''
                add a process to the rd_que only
                '''
                if len(rd_que) != 0 and len(exec_proc) != 0:
                    for k in range(len(rd_que)):
                        if rd_que[k][0] == exec_proc[len(exec_proc) - 1]:
                            rd_que.insert((len(rd_que) - 1), rd_que.pop(k))
                '''
               Appended recently executed process behind rd_que
                '''
            elif proc_data[i][3] == 0:
                tem.ext([proc_data[i][0], proc_data[i][1], proc_data[i][2], proc_data[i]
[4]])
                nor_que.append(temp)
                temp = []
        if len(rd_que) == 0 and len(nor_que) == 0:
            break
        if len(rd_que) != 0:
            if rd_que[0][2] > t_s:
                time_St.append(s_time)
                s_time = s_time + t_s
```

```
            e_time = s_time
            time_exited.append(e_time)
            exec_proc.append(rd_que[0][0])
            for j in range(len(proc_data)):
               if proc_data[j][0] == rd_que[0][0]:
                  break
            proc_data[j][2] = proc_data[j][2] - t_s
            rd_que.pop(0)
         elif rd_que[0][2] <= t_s:
            '''
            complete exec if burst time is lesser
            '''
            time_St.append(s_time)
            s_time = s_time + rd_que[0][2]
            e_time = s_time
            time_exited.append(e_time)
            exec_proc.append(rd_que[0][0])
            for j in range(len(proc_data)):
               if proc_data[j][0] == rd_que[0][0]:
                  break
            proc_data[j][2] = 0
            proc_data[j][3] = 1
            proc_data[j].append(e_time)
            rd_que.pop(0)
      elif len(rd_que) == 0:
         if s_time < nor_que[0][1]:
            s_time = nor_que[0][1]
         if nor_que[0][2] > t_s:
            '''
            If burst time is greater then switch
            '''
            time_St.append(s_time)
            s_time = s_time + t_s
            e_time = s_time
            time_exited.append(e_time)
            exec_proc.append(nor_que[0][0])
            for j in range(len(proc_data)):
               if proc_data[j][0] == nor_que[0][0]:
                  break
            proc_data[j][2] = proc_data[j][2] - t_s
         elif nor_que[0][2] <= t_s:
            '''
            burst time less than time slice, complete execution
            '''
            time_St.append(s_time)
            s_time = s_time + nor_que[0][2]
```

```
                e_time = s_time
                time_exited.append(e_time)
                exec_proc.append(nor_que[0][0])
                for j in range(len(proc_data)):
                    if proc_data[j][0] == nor_que[0][0]:
                        break
                proc_data[j][2] = 0
                proc_data[j][3] = 1
                proc_data[j].append(e_time)
        t_time = RoundRobin.calTATime(self, proc_data)
        w_time = RoundRobin.calcWTime(self, proc_data)
        RoundRobin.printData(self, proc_data, t_time, w_time, exec_proc)

    def calTATime(self, proc_data):
        total_turnA_time = 0
        for i in range(len(proc_data)):
            turnA_time = proc_data[i][5] - proc_data[i][1]
            total_turnA_time = total_turnA_time + turnA_time
            proc_data[i].append(turnA_time)
        average_turnA_time = total_turnA_time / len(proc_data)

        return average_turnA_time

    def calcWTime(self, proc_data):
        total_W_time = 0
        for i in range(len(proc_data)):
            WTing_time = proc_data[i][6] - proc_data[i][4]
            '''
            WTing_time = turnA_time - burst_time
            '''
            total_W_time = total_W_time + WTing_time
            proc_data[i].append(WTing_time)
        average_WTing_time = total_W_time / len(proc_data)

        return average_WTing_time

    def printData(self, proc_data, average_turnA_time, average_WTing_time, exec_pr
oc):
        proc_data.sort(key=lambda x: x[0])
        '''
        Sort processes according to the Process ID
        '''
        print("Proc_ID  Arrival_Time  Rem_Burst_Time  Completed  Original_Burst_Tim
e  Completion_Time  TurnA_time  WTing_time")
```

```
    for i in range(len(proc_data)):
        for j in range(len(proc_data[i])):
```

Endnotes

[1]. Q. Liu, T. Han & E. Moges, *"EdgeSlice: Slicing Wireless Edge Computing Network with Decentralized Deep Reinforcement Learning". In Proceedings of the 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), Singapore*, **2020**, pp 234–244.

[2]. D.M. Casas-Velasco, O.M.C. Rendon, & da Fonseca, ***"N.L.S. DRSIR: A Deep Reinforcement Learning Approach for Routing in Software-Defined Networking". IEEE Trans. Netw. Serv. Manag.***, 2021, pp 1–14.

[3]. A. Dixit, H. Fang, S. Mukherjee, T.V. Lakshman & R.R. Kompella, *"ElastiCon: An Elastic Distributed SDN controller," Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ACM*, **2014**, pp 17-28.

[4]. X. Yang, Y. Wang, I. Wong, Y. Liu & L. Cuthbert, *"Genetic Algorithm in Resource Allocation of RAN Slicing with QoS Isolation and Fairness". In Proceedings of the 2020 IEEE Latin-American Conference on Communications (LATINCOM), Santo Domingo, Dominican Republic*, **2020**, pp 1–6.

[5]. B. Xiang, J. Elias, F. Martignon & E. Di Nitto, *"Joint Network Slicing and Mobile Edge Computing in 5G Networks". In Proceedings of the ICC 2019-2019 IEEE International Conference on Communications (ICC), Shanghai, China*, **2019**, pp 1–7.

[6]. J. Du, C. Jiang, H. Zhang, Y. Ren & M. Guizani, ***" Auction Design and Analysis for SDN-Based Traffic Offloading in Hybrid Satellite-Terrestrial Networks," IEEE Journal on Selected Areas in Communications***, **2018**, pp 2202-2217.

[7]. J. Biswas, A. A. Lazar, J. F. Huard, K. Lim, S. Mahjoub, L. F. Pau, M. Suzuki, W. Wang, & S. Weinstein, ***"The IEEE P1520 Standards Initiative for Programmable Network Interfaces," Comm. Mag.***, **1998**, pp 64–70.

Chapter Four

Results and Discussion of Findings

4.1 Scenario Discussions

This chapter describes the proposed scenarios, then shows and explains the results obtained with the scenarios proposed.

4.2 Network Topology

A three-level fat-tree Data Center topology was selected as the network topology for this study. Eight servers, four edge switches, four aggregation switches, and two core switches make up this system. As shown in Figure 4-1.

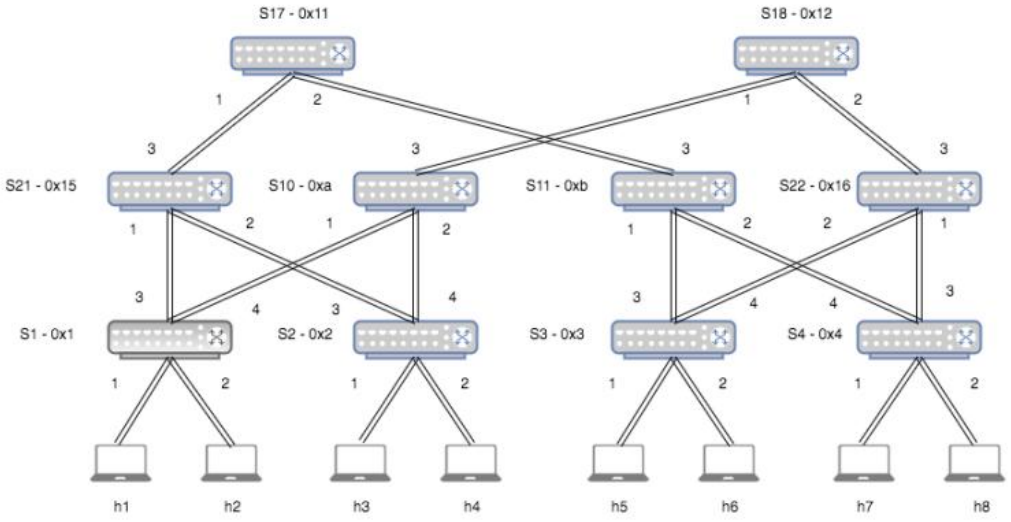

Figure 4.1 Data Center Network Topology Used

(Source: Research Design 2022)

4.3 Scenario Description

4.3.1 First Scenario: Performance Measurement at the Aggregation Layer

The servers' h1 and h4 have been chosen in this scenario to perform load balancing between them. The figure 4-2 below illustrates.

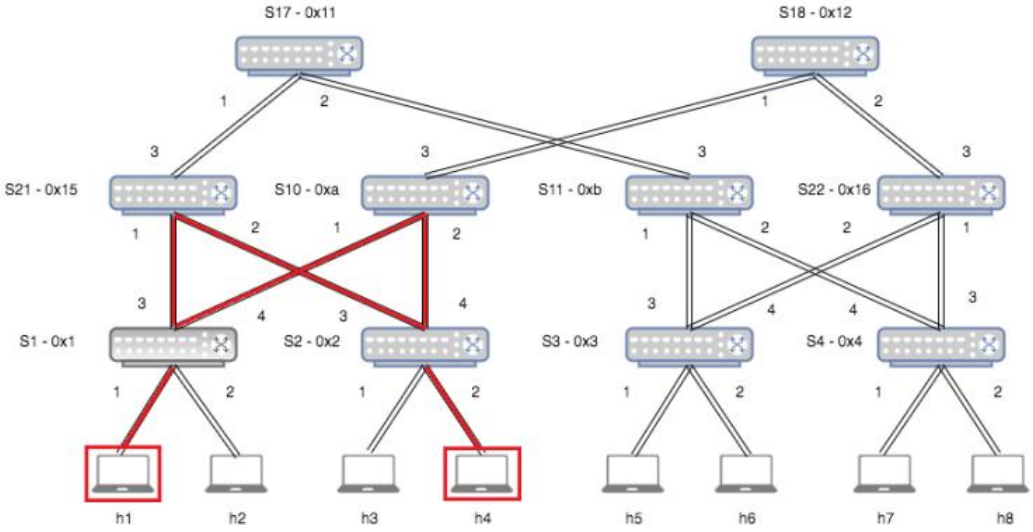

Figure 4.2: The Selected Hosts and Possible Paths in the First Scenario

(Source: Research Design 2022)

Prior to and following the load balancing algorithm's execution, the network was tested. Throughput, delay, jitter, and packet loss between the two servers in the fat-tree network were among the QoS parameters that were tested. The amount of time it took for the source host to receive an ICMP Echo Reply after sending five ICMP Echo Request packets to the destination

host was used to compute the delay. Using iPerf, throughput, jitter, and packet loss were examined for 10 seconds for each test using TCP and UDP, respectively.

The following figures 4.3 to 4.5 show examples of the testing results.

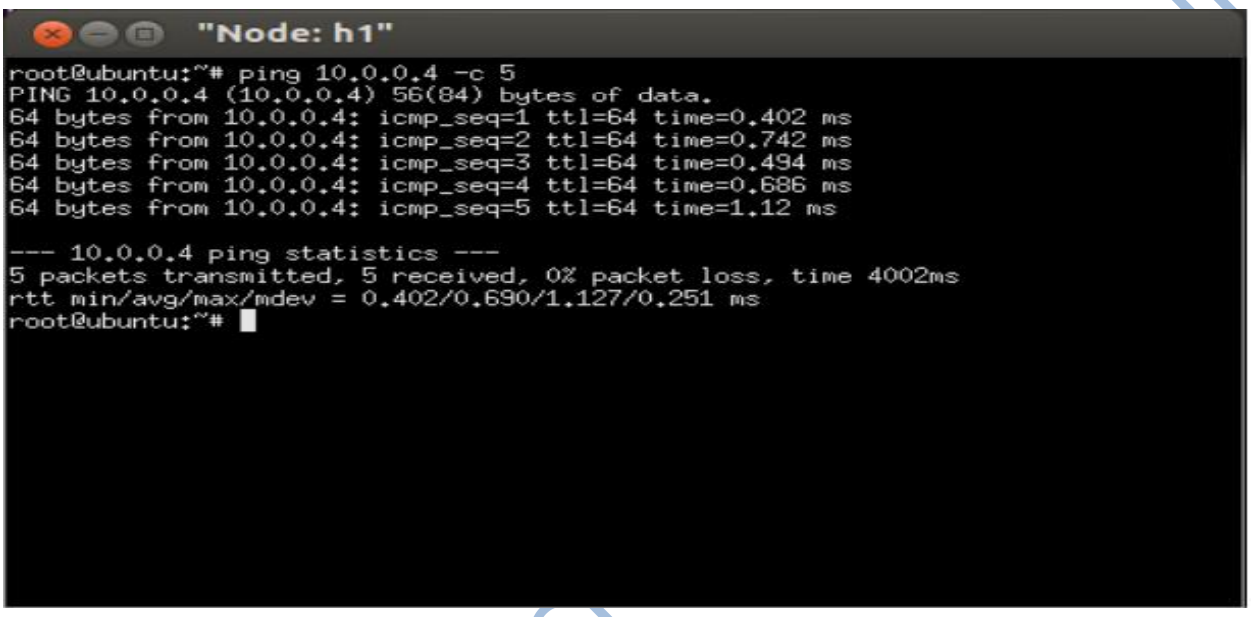

Figure 4.3: Ping from H1 to H4 before Load Balancing

(Source: Research Design 2022)

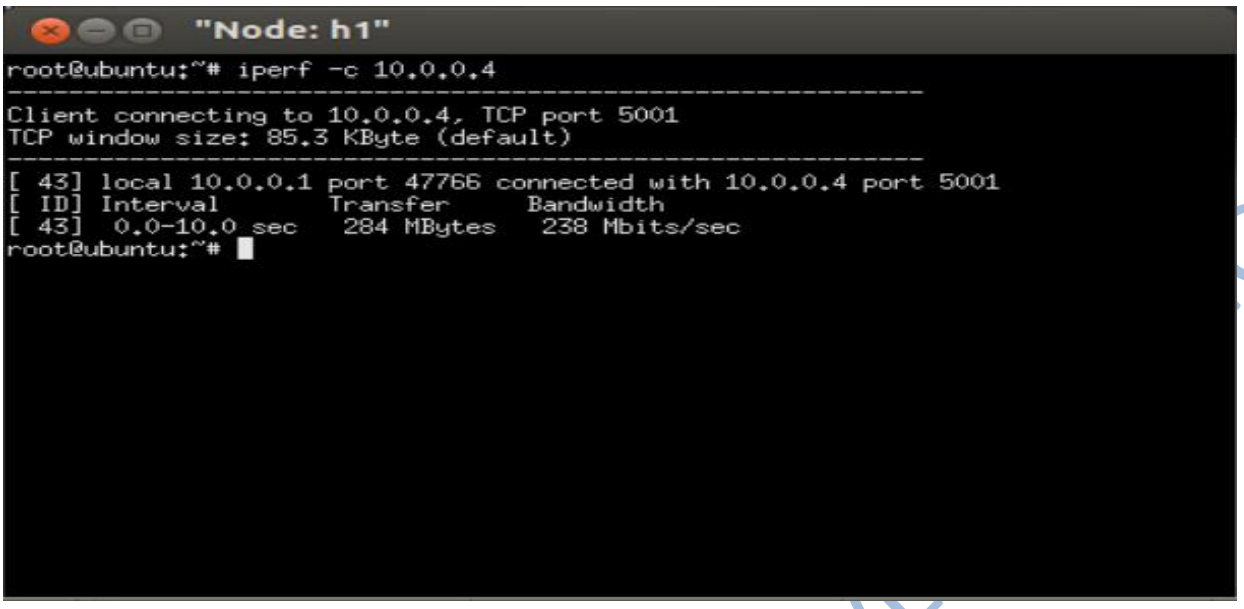

Figure 4.4: IPerf H1 to H4 before Load Balancing – TCP Connection

(Source: Research Design 2022)

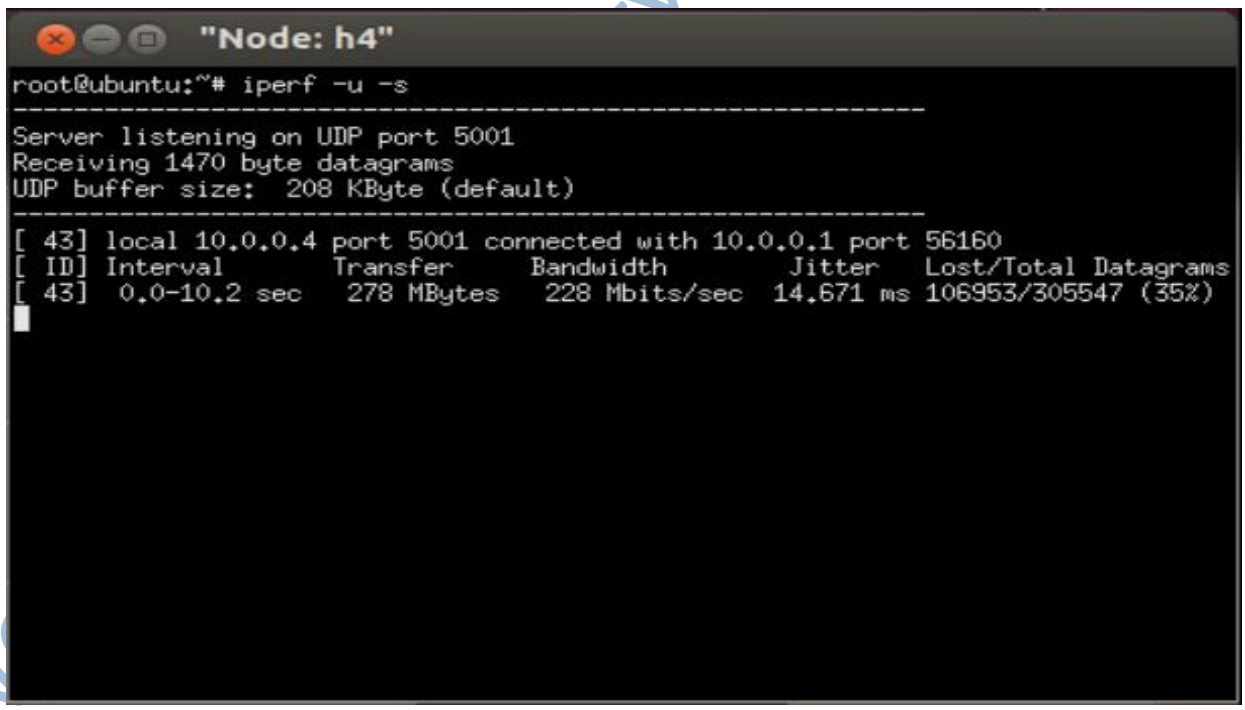

Figure 4.5: IPerf H1 to H4 before Load Balancing – UDP Connection

(Source: Research Design 2022)

4.3.1.1 Tests Results of the First Scenario

Before and after the load balancing algorithm was applied, the network was checked ten times to look for any unusual activity. The results are shown in the following table.

Table 4.1: Tests Results of the First Scenario

Test No.	Load Balancing	TCP		UDP				
		Throughput (Mbits/sec)	Transfer (Mbytes)	Throughput (Mbits/sec)	Transfer (Mbytes)	Jitter (ms)	Packet Loss %	Delay (ms)
1	Before	191	228	354	422	0.641	15%	0.297
	After	13414.4	15564.8	524	594	0.081	13%	0.142
2	Before	221	265	299	357	0.325	38%	0.496
	After	28262.4	32870.4	726	866	0.011	5%	0.176
3	Before	255	304	274	327	0.521	59%	0.203
	After	16076.8	18739.2	713	849	0.006	6%	0.09
4	Before	185	220	361	431	0.491	52%	0.578
	After	21196.8	24678.4	765	912	0.001	2%	0.101
5	Before	238	284	228	278	14.67	35%	0.69
	After	28876.8	33689.6	653	778	0.005	11%	0.159
6	Before	184	223	283	338	0.42	41%	0.529
	After	26828.8	31334.4	625	745	0.176	12%	0.16
7	Before	208	249	256	305	0.365	30%	0.518
	After	30617.6	35635.2	743	885	0.014	4%	0.133
8	Before	233	278	298	355	0.543	36%	0.425
	After	34201.6	39833.6	738	880	0.013	5%	0.115
9	Before	236	281	255	304	0.691	43%	0.707
	After	24268.8	28262.4	742	885	0.193	4%	0.152
10	Before	244	292	215	256	0.172	42%	0.708
	After	36761.6	42803.2	740	881	0.015	3%	0.144

To summarize the previous table, an average performance has been calculated as shown in the following table.

Table 4.2: Average Results of the First Scenario.

Load Balancing	TCP		UDP				Delay (ms)
	Throughput (Mbits/sec)	Transfer (Mbytes)	Throughput (Mbits/sec)	Transfer (Mbytes)	Jitter (ms)	Packet Loss %	
Before	219.5	262.4	282.3	337.3	1.884	39.10%	0.5151
After	26050.56	30341.12	696.9	827.5	0.0515	6.43%	0.1372

4.3.1.2. Performance Analysis of the First Scenario

The network showed a much better performance in the first scenario after running the load balancing program. The average network Throughput before load balancing was 219.5Mbits/sec, and it became 25.4Gbits/sec after load balancing. The average delay has decreased by 73.36% after load balancing with an average of 0.1372ms, the Jitter has decreased by 97.27%, and the Packet Loss has decreased by 32.67%.

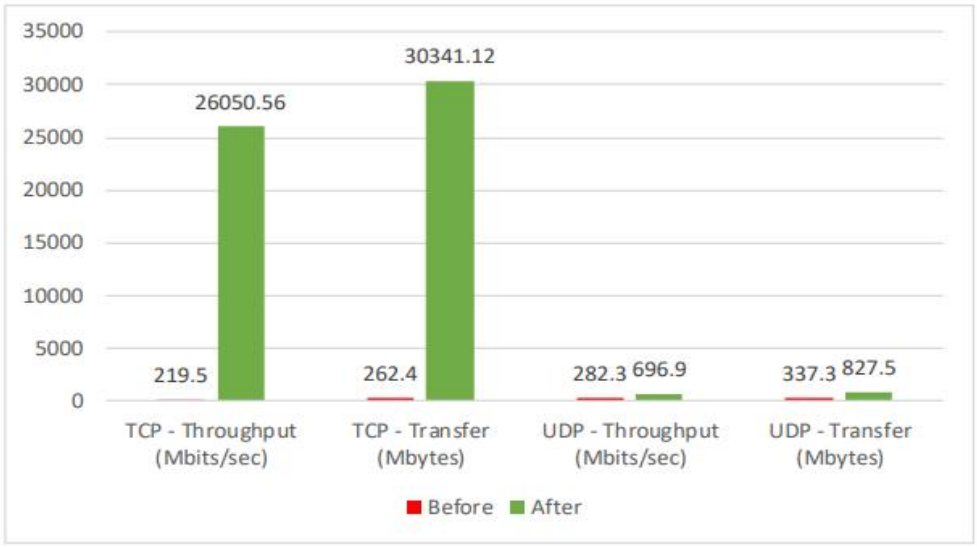

Figure 4.6: Comparison of Throughput Tests Results in First Scenario.

(Source: Research Design 2022)

Figure 4.7: Comparison of QoS Parameters in First Scenario.

(Source: Research Design 2022)

4.3.2 Second Scenario: Performance Measurement at the Core Layer

In this scenario the server's h1 and h6 has been selected to perform the load balancing between them. In this scenario the traffic will have to go through the core switches in order to reach its destination. The figure 4.8 below shows the selected hosts and the possible paths.

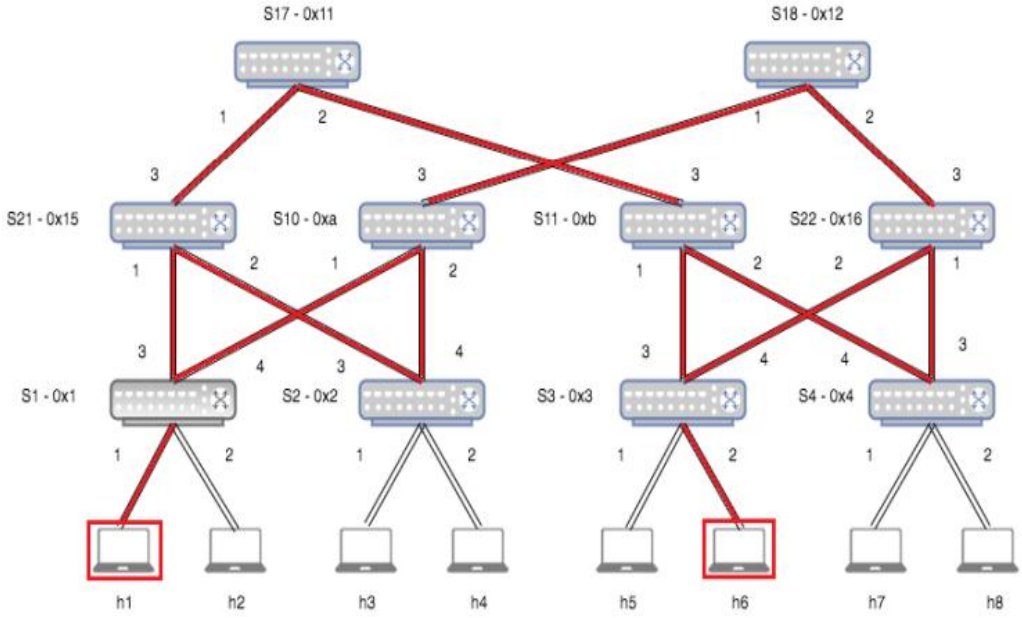

Figure 4.8: The Selected Hosts and Possible Paths in the Second Scenario.

(Source: Research Design 2022)

The network was tested before and after running the load balancing algorithm. The testing focused on some of QoS parameters such as throughput, delay, jitter, and packet loss between the two servers in the fat-tree network.

The following figures 4.9 to 4.11 show examples of the testing results.

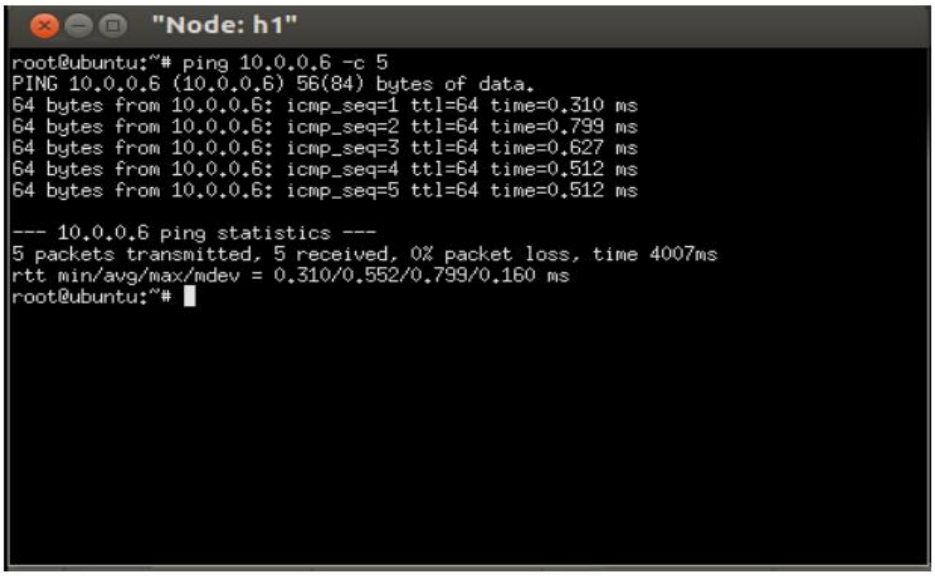

Figure 4.9: Ping from H1 to H6 before Load Balancing

(Source: Research Design 2022)

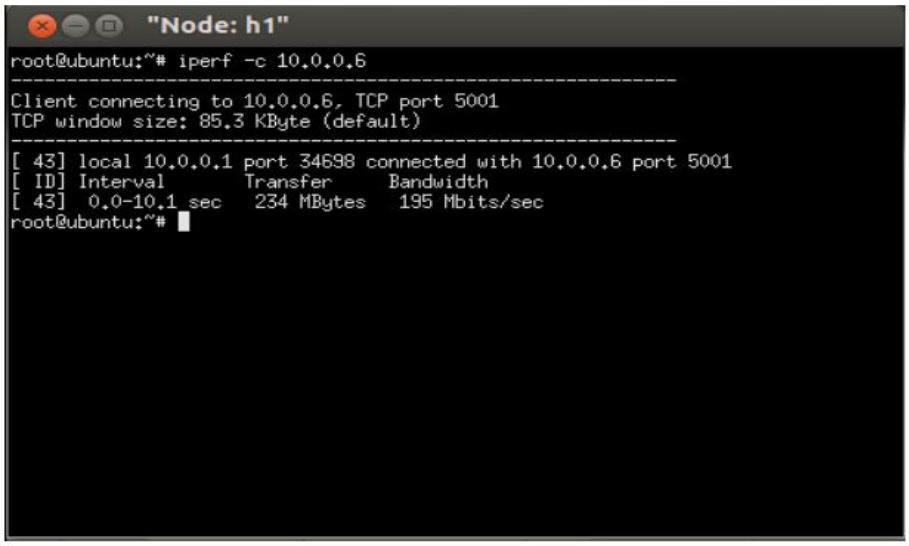

Figure 4.10: IPerf H1 to H6 before Load Balancing – TCP Connection

(Source: Research Design 2022)

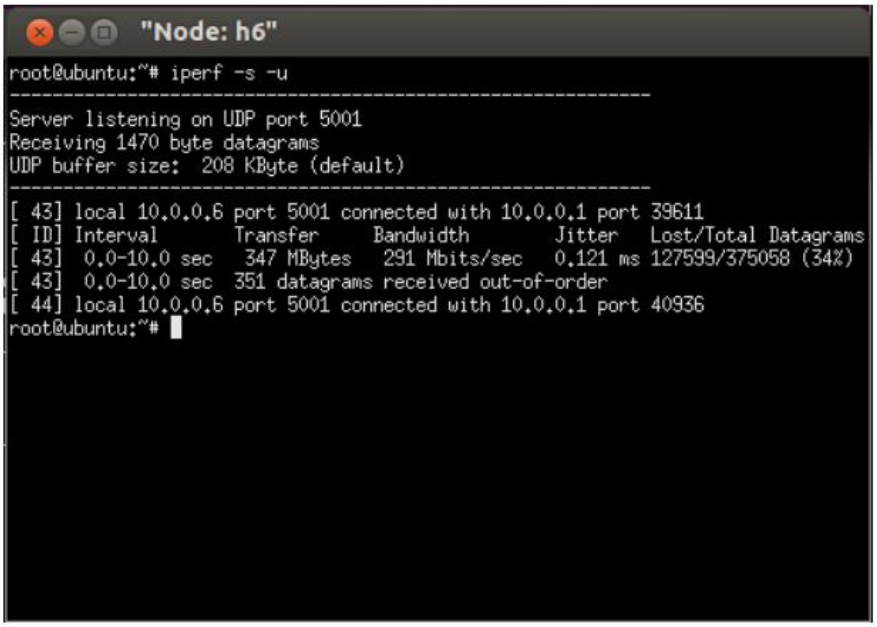

Figure 4.11: IPerf H1 to H6 before Load Balancing – UDP Connection

(Source: Research Design 2022)

4.3.2.1 Tests Results of the Second Scenario

Before and after the load balancing algorithm was applied, the network was checked ten times to look for any unusual activity. The results are shown in the following table.

Table 4.3: Tests Results of the Second Scenario.

Test No.	Load Balancing	TCP		UDP				Delay (ms)
		Throughput (Mbits/sec)	Transfer (Mbytes)	Throughput (Mbits/sec)	Transfer (Mbytes)	Jitter (ms)	Packet Loss %	
1	Before	195	234	291	347	0.121	34%	0.552
	After	268	319	263	313	0.438	51%	0.439
2	Before	194	233	225	268	0.061	54%	0.268
	After	273	326	336	401	0.529	52%	0.323
3	Before	150	179	147	176	0.889	39%	0.943
	After	215	257	203	242	0.529	42%	0.561
4	Before	183	219	201	239	0.292	34%	0.374
	After	240	287	238	288	14.12	44%	0.401
5	Before	187	225	194	231	0.564	45%	0.242
	After	236	283	232	283	14.32	36%	0.662
6	Before	219	261	226	268	0.166	36%	0.553
	After	288	344	380	453	0.567	51%	0.618
7	Before	203	243	246	293	0.691	46%	0.566
	After	222	266	367	437	0.537	53%	0.499
8	Before	164	196	225	268	0.128	28%	0.596
	After	220	263	252	299	0.587	35%	0.432
9	Before	220	264	258	307	0.415	38%	0.615
	After	301	359	281	335	0.335	38%	0.332
10	Before	176	210	179	213	0.409	40%	0.582
	After	239	288	234	280	0.913	41%	0.54

To summarize the previous table, an average performance has been calculated as shown in the following table.

Table 4.4: Average Results of the Second Scenario.

Load Balancing	TCP		UDP				Delay (ms)
	Throughput (Mbits/sec)	Transfer (Mbytes)	Throughput (Mbits/sec)	Transfer (Mbytes)	Jitter (ms)	Packet Loss %	
Before	189.1	226.4	219.2	261	0.3736	39.40%	0.5291
After	250.2	299.2	278.6	333.1	3.2891	44.30%	0.4807

4.3.2.2. Performance Analysis of the Second Scenario

In the second case, the load balancing program was performed, and the network displayed good performance. The average network throughput was 189.1Mbits/sec, but after load balancing, it increased by 32.3% to 250.2Mbits/sec. After load balancing, the average delay has decreased by 9.15%, to an average of 0.4807 milliseconds. However, following the load balancing, the average jitter increased from 0.3736ms to 3.2891ms, and the packet loss also increased by 4.9%. The throughput was always increased by the load balancing application. Under the second layer of the fat-tree topology, it performed admirably, but as the network gets bigger and the core layer is involved, it exhibits growing jitter and packet loss.

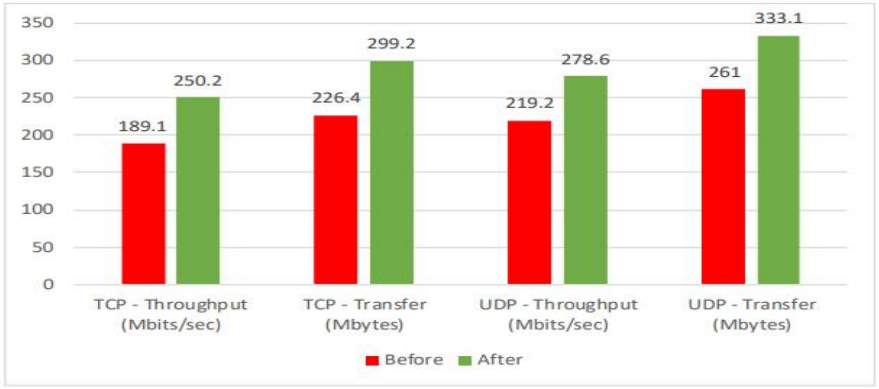

Figure 4.12: Comparison of Throughput Tests Results in Second Scenario

(Source: Research Design 2022)

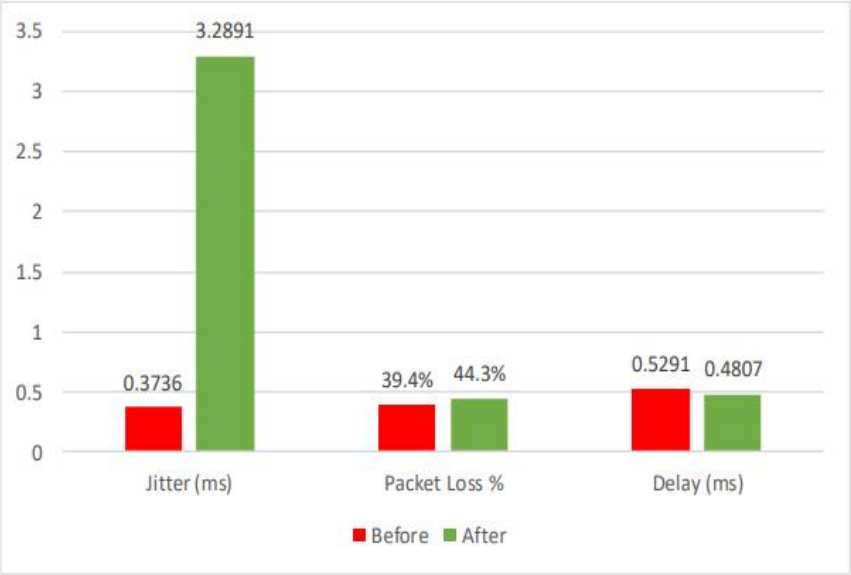

Figure 4.13: Comparison of QoS Parameters in Second Scenario

(Source: Research Design 2022)

Chapter Five

Conclusion

5.1 Summary of Findings

This study outlines the use of certain load balancing techniques and algorithms to effectively distribute flows for fat-tree networks via numerous alternate routes between a single pair of hosts. Both before and after the load balancing method was applied, the network was tested. Throughput, latency, and packet loss between two servers in the fat-tree network were among the QoS metrics that were tested. The load balancing method was able to boost throughput and improve network usage, and the findings demonstrated that the network performance had improved after the algorithm had been performed. However, it caused more packet loss and jitter in large networks.

5.1.1 Conclusions

In laboratory conditions, the test scenario are performed in which the testbed environment was composed of a minimum number of devices needed to realize the project objectives. This approach produces a particular limit in terms of results - they could be different if it the testing were performed in a realistic or cloud environment. As software-defined network is developed to manage large networks like WAN, cloud computing technologies like data center, big data etc. Growth in today's network leads to large amount of traffic on the link due to which performance and efficiency of the network degrades. The algorithm is not analyzed over such a big network. One can check the performance on these networks and update the algorithm according to the experimental results.

5.2 Contribution to Knowledge

Software-defined networking (SDN) load balancing removes the protocols at the hardware level to allow for improved network management and diagnosis. SDN controller load balancing makes data path control decisions without having to rely on algorithms defined by traditional network equipment. An SDN-based load balancer saves running time by having control over an entire network of application and web servers. Load balancing in SDN leads to discovery of the best pathway and server for the fastest delivery of requests.

5.3 Suggestions for Further Studies

Future research and approaches to load balancing are discussed.

1. Developing additional functionalities based on SDN technology, which will improve traditional computer networks' functioning and solve problems, such as IP mobility in heterogeneous networks, more efficiently. In that sense, special attention will be directed towards developing new weights distribution models for coefficients, considering the previously mentioned fact.
2. More precise identification of the impact of the proposed load balancing scheme on low latency traffic.
3. Evaluate the outcomes after examining the dynamic load balancing program's performances on a variety of well-known SDN controllers, like Research Floodlight, Beacon, NOX/POX, etc.
4. Examine how well topologies other than the fat-tree topology of various sizes perform. To see if the algorithm has any further restrictions.