SOFTWARE BASED LOAD BALANCING TO ENHANCE NETWORK PERFORMANCE

AZLINDA BINTI AZMAN

SOFTWARE BASED LOAD BALANCING TO ENHANCE NETWORK PERFORMANCE

AZLINDA BINTI AZMAN

This report is submitted in partial fulfilment of the requirements for the Bachelor of Computer Science (Computer Networking)

FAKULTI TEKNOLOGI MAKLUMAT DAN KOMUNIKASI

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

2021

DEDICATION

To my beloved parents, thank you for bringing me into this world as a good human being.

To all my friends who always support me in completing this project.

Thank you so much.

Lastly, special thanks for my supportive supervisor, thank you for all the motivation and for believing in me completing this project.

May we all be blessed.

ACKNOWLEDGEMENT

Praise to the All Almighty, Allah SWT for giving me the opportunity and strength to finish my Final Year Project. I would like to thank Dr Zaheera Zainal Abidin for her time assisting me in completing this project successfully.

To my family, I would like to express my gratitude for their support and understanding towards me throughout my journey in completing my project. Thank you to my mother Wan Rizah Binti Shahid for inspiring me with your wonderful energy to keep me working hard to complete this project.

To all of my friends that never stop giving advice and support towards me while completing this project, big gratitude to you all. Hope everything will be good for you too.

Last but not least, I am grateful to Universiti Teknikal Malaysia Melaka (UTeM) for providing me with this opportunity.

ABSTRACT

Software Defined Networking (SDN) is an architectural network approach that allows software-based load balancer applications to control, programme and manage the network. SDN separates the network configuration and data traffic from the root hardware infrastructure, to ensure integrated and harmonious control of the network. The SDN enables network behaviour to be programmed in a centralised manner using APIs.

Furthermore, users programme, manage the entire network and implement network devices consistency by adopting a common SDN layer while disregarding the complexity of the root network technology in the legacy network. In this project, SDN offers a lot of benefits to find. SDN centralised network provisioning, allowing for enterprise management and provisioning to be centralised. More VLANs, for example, are being created and part of the physical LANs, which produce links of Gordian Knot and dependencies. As a matter of fact, SDN offers benefits such as lower operating cost, administrative efficiency, improvements in server utilization and a better control of virtualization.

However, SDN consists of limitations such as security problems and managing load balancing. Security and load balancing are two common problems in SDN since hardware routers and switches have been removed from the network, and security features such as firewalls are no longer available in the network architecture. These limitations and disadvantages of SDN are discussed and examined in this final year project to ensure that SDN offers ways that are widely used in the future. Therefore, the SDN approach is simulated using Mininet, and OpenDayLight to evaluate the characteristics and performance of the security and load balancing to be operated on the network.

TABLE OF CONTENTS

LIST OF FIGURE

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Introduction

Software-Designed Networking is an architectural network approach that allows the network to be controlled, programmed and managed by software applications. SDN separates the network configuration and traffic from the root hardware infrastructure, to ensure integrated and harmonious control of the network. In this project, a simulation-based research has been done using SDN using GNS3.

Generally, SDN permits the programming network manners in a centrally controlled manner through software applications using APIs. By initiating traditionally closed network platforms and implementing a common SDN layer, users can manage the whole network and network devices consistency, ignoring the complexity of the root network technology.

1.2 Problem Statement

A SDN is a software-based network controller producing flexible, scalable, cost effective and adaptive features that is ideal for high-bandwidth and complex applications. However, due to the increasing demands for internet bandwidth from the new internet-of-things devices and mobile users, the internet is still not sufficient to meet the high network requirements, such as flow requests, load balancing and security.

The traditional load balancer is a vendor manufactured on specific hardware development which is costly, inflexible and non-programmable to use. Thus, network administrators are unable to customize the settings and cannot create their own algorithm. To overcome this problem, this project emulates scenarios of network controller technologies.

Table 1.1 Summary of Problem Statement (PS)

PS 1	The traditional load balancer is unable to be customized and non-programmable.
PS 2	The SDN based load balancing offers a visualization platform.
PS 3	The variation of performance in SDN based Load Balancing.

1.3 Project Question

Table 1.2 Summary of Problem Question

PQ 1	How is load balancing customised and programmed in the existing network?
PQ 2	How to perform visualization on SDN based load balancing?
PQ 3	What is the difference between having a load balancer or without it affecting the SDN performance?

1.4 Project Objective

Nowadays, an enormous usage of bandwidth for data streaming and an increase on the use of Internet-of-Things (IoT) technology on various devices world-wide demand. Software Defined Networking (SDN) to be the software-based controller to be used widely in improving the administrative efficiency, server utilization, a better control of virtualization and reducing the operating cost.

Thus, in this project, a study is carried out on SDN and how it can be applied in the legacy Networking environment. Moreover, investigation on the advantages and the disadvantages of the SDN is carried out to make sure this SDN is one of the software-based controllers used in the networking environment. SDN has several drawbacks, which is security and Load Balancing, but in this study the load balancing is under study. Therefore, in this project, the SDN is examined using the simulation approach using GNS3 emulator. The objective of the study is summarized in Table 1.3.

Table 1.3 Summary of Project Objective

PO 1	To study Software-Defined Network (SDN) in the legacy network environment.
PO 2	To implement the SDN using open-source tools GNS3, Mininet and OpenDayLight.
PO 3	To analyze SDN performance based on several algorithms in the load balancing controller.

1.5 Project Scope

The scope of this study is to use the GNS3 as an SDN simulator. GNS3 is a network simulator similar to Packet Tracer but with advanced technology developed. GNS3 is a network software emulator that combines virtual and real-world devices to simulate complicated networks. Computer network simulation is a significant modern technology that allows for quick and cost-effective network testing and validation.

1.6 Project Contribution

Table 1.4 Summary of Project Contribution

PS	PQ	PO	PC	Description
PS1	PQ1	PO1	PC1	A study on a new load balancer using SDN that customised the new legacy of the network environment.
PS2	PQ2	PO2	PC2	Implementation of SDN using open-source tools such as GNS3, Mininet and OpenDayLight.
PS3	PQ3	PO3	PC3	Analysis of SDN performance based on several algorithms in the load balancing controller.

1.7 Report Organization

The final year project report consists of seven chapters, which ~~will~~ include Introduction, Literature Review, Methodology, Analysis and Design, Implementation, Testing and Conclusion.

This project's introduction is covered in Chapter 1. This chapter contains a problem statement as well as project objectives. The project's background on Software Defined Network is covered in Chapter 1.

The literature review in Chapter 2 outlines previous research that were conducted before this approach was offered to the community. Using the studies as a reference allows for a better understanding of the project.

The methodology of the project is discussed in Chapter 3. This chapter explains how GNS3 was utilised to build SDN. This will be the use of software in which the project will be carried out. The software will be explained in great detail.

The fourth chapter deals with project analysis and design. There is a need for analysis and design. It is concerned with the project's approach, whereas chapter five is concerned with the experimental. The process of creating a system prototype is well-documented. It's an important aspect since it confirms the project's efficacy once it's finished.

The fifth chapter deals with the implementation of the project which consists of software setup and configuration.

Chapter 6 involves a Testing process where simulation will be done using GNS3 to test out the Load Balancing in SDN approach.

The final chapter of the project is Chapter 7. Also, share future ideas for improving the present prototype.

1.8 Conclusion

The expected objective of this project is that it will illustrate how simulation of various network scenarios can be developed and how it may affect network performance. Furthermore, this study evaluated the efficiency of the load balancing actuators and compared the latency and reaction time for the various design topologies. The objective of this project is to create a load balancing network scenario utilising network controller software tools.

CHAPTER 2: LITERATURE REVIEW

2.1 Software Defined Network

SDN (Software Defined Networking) is a new networking model that makes network management activities even easier. Furthermore, it allows for network evolution by providing a programmable flexible interface that controls the whole network's action. Classic IP networks, on the other hand, have been difficult to handle, error-prone, and difficult to implement new functionalities for decades. Traditional IP network protocols were created with a distributed control architecture in mind, requiring network devices to connect with one another using a vast number of network protocols in order to negotiate the exact network activity depending on each device's configuration. Network computers are marketed as closed modules, and network operators may only modify the parameters of various network protocols. Network administrators can convert high-level network rules into low-level scripts written for each specific system, referred to as the "Configuration Language."

The networking market has undergone a paradigm change as a result of SDN. Rather than using a hierarchical control architecture, it concentrates all control in a single node known as the "Network Controller," which is essentially software operating on a commercial cloud platform. Network forwarding machines are no longer involved in network management and only forward packets according to a series of rules installed by the network controller. The “Openflow protocol” is used by the network controller to programme the forwarding rules of the forwarding devices, and hence the network forwarding devices are referred to as “Openflow switches.” Since Openflow is a vendor-independent standard protocol, no special knowledge

of the equipment vendor is needed to monitor the forwarding actions.The Openflow protocol was created in late 2008 by a group of researchers from various universities, including Stanford University, the University of Washington, MIT, Princeton University, and others.

For the development of new services, SDN has created a very scalable gui. Network programmers must use a high-level programming language to create their own network policies and facilities (one example of network policies is load-balancing the traffic to a certain destination over multiple paths to avoid the congestion of a certain path). These high-level programmes should be able to be translated into low-level forwarding laws that can be applied to individual forwarding devices by the network controller. Network managers can simplify network activities by using unified high-level programmes to monitor network activity. These programmes are written in high-level general-purpose programming languages such as C++, Java, and Python.

2.1.1 Software Defined Network Architectures

SDN architectures are divided into 4 layers. The first layer is the Network Data Processor. There are data processing machines in this layer. The activities installed by the network controller in each individual system determine how data packets are managed. The basic protocol used to communicate with the network controller and forwarding devices in order to instal data processing rules is known as Openflow. Priority, matching state, operation, and associated active counters are the four key sections of an openflow law. The priority field is used to specify the order in which data packets are matched against the installed rules; if a higher priority rule is matched, lower priority rules are ignored. Any mixture of IP/Ethernet headers (such as source IP, destination IP, port numbers, and VLAN id) may be used as a matching condition. The operation field specifies what can be done with the packet after it has been sent, such as forwarding it to an outgoing interface, modifying any of the header fields before forwarding, or dropping it. Finally, the counters sector specifies the corresponding counters for this law, such as the count of the total of matching packets, so that the values can be sent to the controller for data plane statistics. Openflow has undergone several changes, including the inclusion of several additional matching criteria and behaviours to accommodate more complex use cases.

Layer 2 is known as Network Operating System. The network control is present in this layer, which is why it is referred to as the "Network Controller" or "Network Brain." Based on the logic specified by the running applications, a controller interacts with either the data forwarding devices to load, upgrade, and uninstall openflow standards. The SDN controller performs important functionality to the SDN applications, similar to how a typical computer's

operating system performs functions like resource management and file system management. There are several open source SDN controllers that bring different openflow versions, programming interfaces, and, ultimately, various utilities for running applications. The controller has many durability, scalability, security, and performance issues due to its centralised design. There are no specifications specifying the interfaces or utilities provided by the SDN controller to network applications, so there are many difficulties in this layer.

Next, Layer 3 which is the Network Compiler. Network programmes are written in this layer using SDN-specific programming languages, with related compilers translate the program to the proper API provided by the network controller or SDN system software. It is widely recommended to use this layer for the following reasons which is Application Portability, High level of network abstraction and Code reusability. Application portability is not possible for existing SDN networks due to the variety of network controllers and related APIs. To ensure device portability between separate controllers, using a network programming language and only using the correct network compiler will suffice. For necessity, this implies that a compiler programme for the targeted network operating system applies, but this is yet another task that most of the common network operating systems' simulator developers can handle. When using a high-level programming language, the writer is responsible for the programming logic while the programming language is responsible for the low-level operations. This means that the developer can only define high-level policies, and the programmer should be able to evaluate these policies and create similar openflow rules to be installed over specific switches without troubling the developer. Network advancement is getting much simpler as a result of this. The Code reusability, while most controllers have a direct programming interface, the low level structure of the exposed programming APIs prevents conflict-free execution of composite code. The concurrent execution of two basic programmes over a NOX controller (one

programme forwards packets from one interface to another interface while the other programme tests web traffic accessing the same interface), as indicated by Frenetic, would result in entirely incorrect performance, and a third programme should be designed to merge the executions of the two programmes.

The fourth layer is called the Cross Layer. This layer uses two interfaces to link the application and infrastructure layers. The infrastructure layer is connected to the infrastructure layer in a downward direction through a south-bound interface. This gui allows controllers to interact with switching systems and access their capabilities. Providing network status and importing packet forwarding rules are among the features. The north-bound interface is used to link programme layers in a northward direction. It provides service access points in the form of an API that provides network status data from switching devices. Since a network domain can include several routers or switches, an east-west interface is present to allow controllers to exchange network details and coordinate decision-making progress. Knowledge on QoS for network implementations aids in the implementation of cross-layer techniques to achieve the desired QoS. A stronger QoS for video conferencing, for example, can be shown by obtaining a bandwidth timetable from an SDN controller.

2.1.2 Advantages of Software Defined Network

In SDN, a centralised and programmable network may be developed that can dynamically deliver in order to meet the changing demands of organisations. In addition, there are the following benefits of SDN.

First of all, it may be programmed directly. The network can be directly programmed since the control operations are separated from the forwarding operations. This allows the

network to be configured programmatically using proprietary or open source automation technologies such as OpenStack, Puppet, and Chef.

Moreover, management that is centralised. Intelligence relating to networks is logically centralised in the SDN controller software such that it established a strong network view that presents to programs and policy algorithms as an one, logical switch.

Furthermore, Agility and Flexibility are delivered. SDN enables enterprises to quickly deploy new applications, services, and infrastructure in order to keep up with changing business goals and objectives.

Lastly, Providing a platform for innovation. SDN also enables businesses to tailor new sorts of applications, services, and business models, thereby generating new income streams and increasing network value.

2.2 Load Balancing

One of the most difficult aspects of computer networking is load balancing. Memory, CPU capacity, network demand, and even delay load can all contribute to this stress. Load balancers are designed to continually spread the workload of all nodes in a distributed system in order to improve system performance and resource usage. This can also help in circumstances when the nodes in the network have greater or less load by avoiding them. Load balancing is the practise of ensuring that the work load is evenly spread among a pool of system nodes or processors so that the ongoing operation may be finished without interruption.

In (SDN) software defined networking, load balancing functions as an aware routing protocol; it is an essential element that aids availability and scalability, resulting in the shortest possible application response time. Millions of individuals are linked to the internet, resulting in increased web traffic, network congestion, and packet losses. So, in order to address this issue, The use of load balancing strategies improves network efficiency.

Load balancing approach that maximises throughput by decreasing response time intervals and decreasing jams. Traditional load balancing networks are not detailed, but Software Defined Networks are significantly more efficient and have a higher performance.

2.2.1 Two types of Load Balancing

In a static method, traffic is divided evenly across the servers. For systems with little load fluctuation, the static method is ideal. This method requires previous knowledge of the system resources in order to ensure that load shifting decisions are not based on the present condition of the system.

In a dynamic method, the lightest server in the system is sought, and load balancing is prioritised appropriately. As a result of the necessity for real-time connection with the network, the system's traffic may grow. In addition, the current system status is used to make load management decisions.

2.2.2 Network Load Balancing

Network load balancing improves the availability and flexibility of mission-critical net servers and applications such as Web based, firewall, File Transfer Protocol (FTP), proxy, virtual private network, and other servers.

If any of the hosts in the cluster die unexpectedly, Network Load Balancing guarantees that traffic is directed to all of the remaining hosts in the network. Up to 32 servers can be added to a Network Load Balancing cluster. This makes it easier to work on crucial operations.

2.2.3 Applications of SDN in Load Balancing

There are several used to disperse incoming load among numerous servers in order to prevent a single server from being overloaded. First, Network virtualization protects physical networks and divides them into several components. Network virtualization enhances network performance and automation when new software components are introduced.

Moreover, the SDN controller assigns topology discovery as a service. It maintains and gives a global perspective of the network. Furthermore, Traffic monitoring offers the necessary data to warn drivers of potential difficulties as well as information important to road engineers such as vehicle count, speed, and occupancy.

Last but not least, beyond the fundamental specification, security augmentation refers to the enhancement of security capabilities and services.

2.2.4 Load Balancing Techniques

2.2.4.1 Balance Flow

According to (H Sufiev and Y Haddad, 2016), this approach is named "BalanceFlow," and it involves a SuperController doing load balancing amongst network controllers (SC). "Balance flow" focuses on system load balancing such that flow-requests are dynamically divided across controllers for quick response.

To improve controller usage, the load on the overloaded controller is dynamically transferred to an appropriate low-loaded controller. This method necessitates that each switch enables a certain flow of service from a subset of controllers. The algorithm's precision is achieved by splitting the switch load among some controllers by each current's source and destination.

2.2.4.2 Hybrid Flow

The second strategy that may be utilised in network controllers is HybridFlow, which consists of separating the controllers into clusters so that each cluster may aid each other and conduct load balancing inside the same cluster.

When all of the controllers in the cluster are full, a request to the SC to lower the number of switches to be operated in the cluster will be submitted. This "local" load balancing strategy helps to lessen the burden on the SC while maintaining overall load balancing.

2.2.5 Works Related Load Balancing

2.2.5.1 Round Robin

Tolerance against trivial defects is provided using this way. A set of identical servers is allocated to work in such a way that they can all provide the same services. Despite the fact that each server has its own IP address, they are all set up to utilise the same domain name. The DNS server keeps track of all IP addresses linked with Internet domain names. When a request for an Internet domain name and its associated IP address is received, all of the addresses are delivered back in a rotating order.

2.2.5.2 Least Connection

The Least Connection technique takes current server load into account. The request is then sent to the server that has served the fewest number of active sessions at the moment. Least Connection (Weighted) Every server is assigned a number, similar to the Weighted Round Robin technique. This is what the load balancer looks for when assigning server

requests. If the number of active connections on two servers is the same, the higher weighted server will get a new request.

2.2.5.3 Agent Based Adaptive Load Balancing

Every server in a pool is assigned an agent, which reports to the load balancer on its current load. This real-time data is used to determine which server should be used to best handle requests. Other strategies, such as Weighted Round Robin and Weighted Least Connection, are employed in conjunction with this one.

2.2.5.4 Chained Failover

This strategy necessitates a specified sequence of servers that must be configured in the chain in which they are present. All requests are routed to the first server in the chain. If the next server in the chain is unable to take any more requests, all requests are routed to the third server, and so on.

2.2.5.5 Weight Response Time

This approach continually receives response time information from servers in order to determine if the server is responding at its quickest in a certain period of time. The next server access request is made to that server. This is meant to ensure that if a server is currently under a lot of pressure and is responding slowly, no new requests will be sent to it. This allows the load to be distributed equally across the available server pool over time.

2.3 Project Software

2.3.1 Mininet

Mininet is a network emulator that produces a virtual network comprising hosts, switches, controllers, and connections. Mininet switches offer OpenFlow for extremely flexible custom routing and Software-Defined Networking, and its hosts run conventional Linux network software. Mininet can be used for research, development, learning, prototyping, testing, debugging, and any other job that would benefit from having a fully functional experimental network on a laptop or other PC.

2.3.2 GNS3

GNS3 (Graphical Network Simulator) is an open source programme that simulates complicated networks as closely as possible to how they operate in real life. All of this is accomplished without the need of network gear such as routers and switches.

This programme provides an easy-to-use graphical user interface for designing and configuring virtual networks. It operates on standard PC hardware and is compatible with Windows, Linux, and MacOS X.

2.3.3 OpenDayLight SDN Controller

The OpenDaylight controller is a Java virtual machine (JVM) that may operate on any operating system and hardware that supports Java. The controller is an implementation of the Software Defined Network (SDN) idea, and it makes use of Maven, OSGi, JAVA, and Rest APIs.

2.3.4 Conclusion

Load balancing can be implemented using one of two methods: hardware-based or software-based. The load balancing platform is built on software and uses software defined networking technologies. SDN technology is a network management strategy that allows for dynamic, programmatically efficient network design to increase network performance and monitoring, making it more like cloud computing than traditional network administration. Furthermore, load balancing techniques might have an influence on network performance.

2.4 Proposed Solution

According to studies, employing a software-based platform is the best approach to accomplish load balancing. Because a software defined network is an open source software that provides dynamic, programmatically efficient network setup in order to increase network performance, it is the ideal approach to execute load balancing methodology.

The dynamic application load balancing strategy was used in this project. This approach works in conjunction with an external server load balancer, calculating the round robin scheduling weight parameter in the load balancer to distribute requests across nodes.

Finally, based on prior journals, Round Robin is the optimal parameter to utilise. This parameter displays the optimal outcome when used with load balancing technology. This approach is characterised as sending requests one by one in a circular way to each server in the queue. When a packet arrives, the next selected server from the list of all servers on the network system becomes available. So that, except for the load on each server, all servers in the database are in the same sequence and do the same amount of loads.

2.5 Conclusion

As a conclusion, this chapter is critical to the project's completion. A literature review is a summary of research on a certain topic as well as responses to relevant research questions. From the preceding study, we can gather all relevant data and sources. We may learn what the optimal approach, approach, attribute, or parameters are for completing this project by reading this chapter. The approach identifies duplicate computing throughout individual runs as well as across many simulators.

CHAPTER 3: METHODOLOGY

3.1 Introduction

The approach employed in this project will be explained in this chapter. This chapter will detail all of the data elements, population structure, and sample strategies utilised in the

interviews for this study. All of the information acquired will be presented in a step-by-step format, including all of the processes covered in this chapter. Finally, this part provides a full overview of the study approach adopted and the data gathering procedure. In addition, the Gantt chart for this project will be shown in this chapter. The Gantt chart depicts how the work is organised and if the project is on time or not. The Gantt chart's purpose is to direct the project's progress.

3.2 Methodology

Scrum, Kanban, Lean, Waterfall, and Six Sigma are just a few examples of methods that may be utilised to build this project. In this project, waterfall methods were used. Waterfall was the original software development technique, originating in the manufacturing and construction industries, where you can't afford to iterate after you've constructed a tower or a bridge since you can't go back and "upgrade" the foundation.

Winston W. Royce introduced the Waterfall model in 1970. It's simple to comprehend and use. In a waterfall design, each stage must be completed before the next one can begin, and the phases do not overlap. The Waterfall model was the first SDLC approach to be used in software development. The several phases of the waterfall model are depicted in the following diagram.

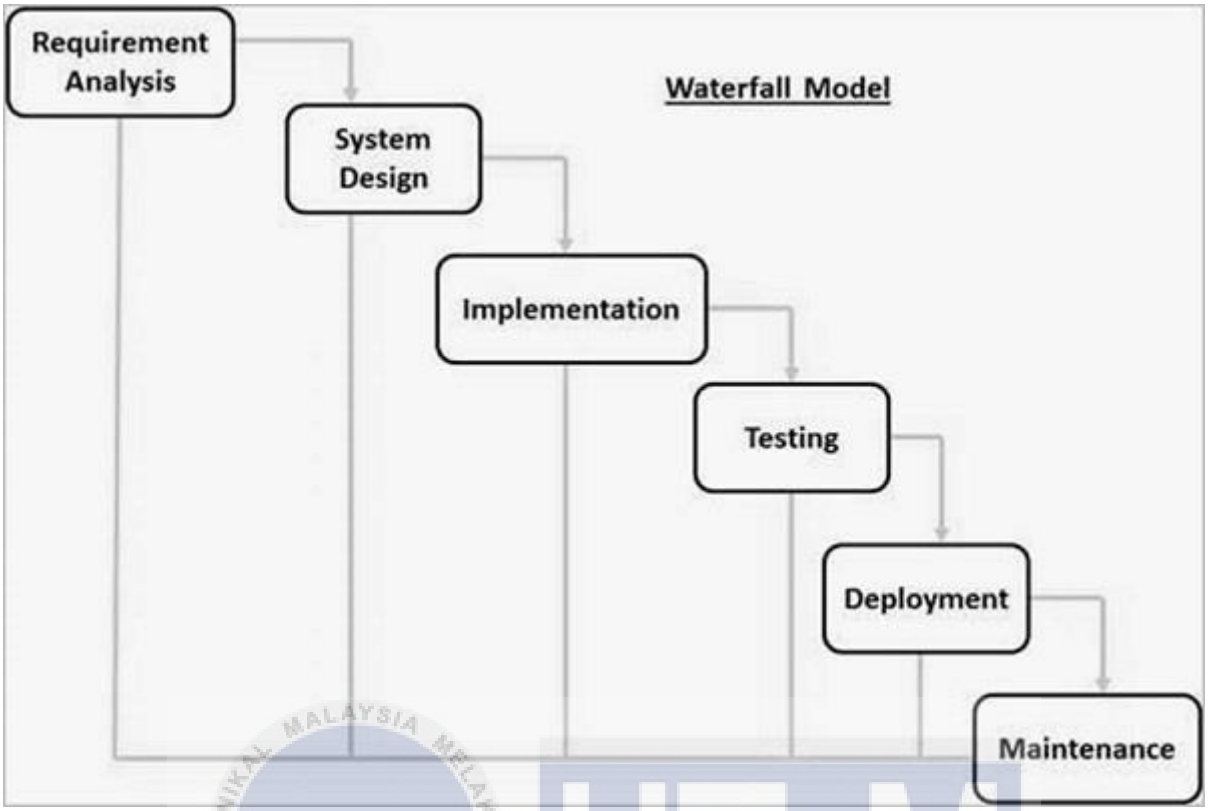

Figure 3.1: Waterfall Model Illustration.

3.2.1 Requirements

This phase captures all potential needs for the system to be created and documents them in a requirement specification document.

3.2.2 Design

This stage examines the requirements specifications from the previous phase and prepares the system design. This system design aids in designing the overall system architecture as well as describing hardware and system requirements. In this project, the designing network structure is a design that must be taken into consideration. A simulation approach employing Mininet software was used to develop the network structure. The network will be designed with a variety of scenarios in mind and load balancing mechanisms will be included

3.2.3 Implementation

The system is first built as tiny programmes called units, using inputs from the system design, in this phase, and then combined in the following step. Unit Testing refers to the process of developing and testing each unit for its functioning. Following the network's design. The network controller and load balancing methods will be used to implement it. The information gathered in chapter two was used to implement this stage.

3.2.4 Testing

After each unit has been tested, all of the units built during the implementation phase are merged into a system. The entire system is then checked for any flaws or failures after it has been integrated. The functionality of each network scenario will be tested, and the findings will be gathered for documenting. The measurement testing, developed with the aim, network control testing, load balancing checking, and added in series The infrastructure was put to the test in terms of throughput, latency, and disturbance.

3.2.5 Deployment

The product is deployed in the client environment or released into the market once functional and non-functional testing is completed.

3.3.6 Maintenance

In the client environment, there are a few challenges that arise. Patches are published to address these vulnerabilities. In order to improve the product, newer versions have been produced. Maintenance is carried out in order to bring about these modifications in the customer's environment. After then, the testing stage will be repeated a few times until a better result is obtained.

3.3 Project Milestones

This project has been assigned project milestones. A milestone is a specific point in the plan's life cycle that was used to evaluate the project's progress toward its final aim. Project Milestones are used to indicate things like the project's start and completion dates, the requirement for external approval or input, financial constraints, the submission of important deadlines, and more. Milestones have a set date but no set duration, ranging from the first week of presentation to the last week of presentation.

Activities / Weeks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FYP Proposal															
Chapter 1 Introduction															
Chapter 2 Literature Review															
Chapter 3 Methodology															
Chapter 4 Design															
Project Demo															
FYP 1 Final Presentation															

Table 3.1 Project Milestones

3.4 Conclusion

In conclusion, this project methodology will consist of 6 phases which are Requirement, Design, Testing, Deployment and Maintenance. This section is critical to this project's success since it ensures that the project follows the correct methodology and that the development process runs smoothly. Gantt charts and project milestones are also critical to ensure that this project stays on track. Design is the next phase to be created. The overall design for this project will be demonstrated in this chapter.

CHAPTER 4: DESIGN

4.1 Introduction

This chapter discuss all of the network design for this project, as well as the outcomes of the preliminary design analysis and the detailed design outcome. This project comprises different network scenarios that will be analysed using simulation tools. The details of each network situation, as well as the parameters of each network architecture, will next be discussed.

4.2 Network Architecture

4.2.1 Software Defined Network Application

A software-defined networking (SDN) application is a programme that performs a job in a software-defined networking environment. It is this approach to computer networking that not only allows network administrators to programmatically adjust, control, initiate, and manage network behaviour through open interfaces, but also introduces the idea of lower-level functionality. SDN applications also aid in the expansion and replacement of tasks that are now performed by firmware in the hardware devices of a traditional network.

4.2.2 Software Defined Network Controllers

To deploy intelligent networks, an SDN controller manages flow control to the switches/routers "below" (through southbound APIs) and the applications and business logic "above" (through northbound APIs). They use common application interfaces to consolidate and mediate between multiple controller domains. SDN controllers communicate with switches/routers via two of the most well-known protocols.

An SDN controller platform usually includes a number of “pluggable” modules that can be used to conduct various network activities. Inventorying what devices are on the network and their capabilities, getting network data, and other monitoring operations are only a few of the essential responsibilities. Extensions that improve functionality and support more advanced features can be added.

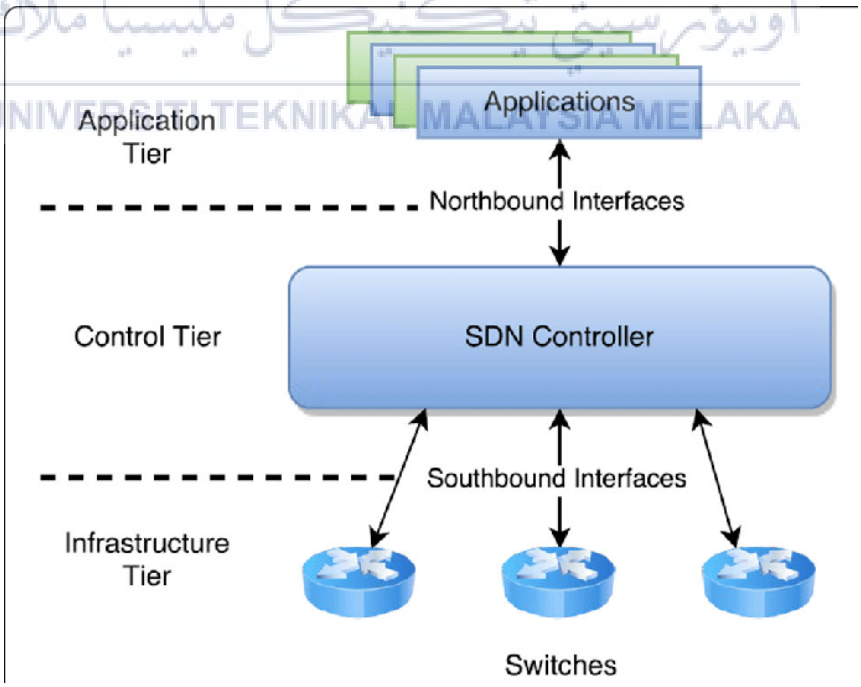

Figure 4.1 Network Architecture of SDN

4.2.3 Experiment Setup of Software Defined Network Controllers

Step 1: Setting up a network environment as shown below on GNS3.

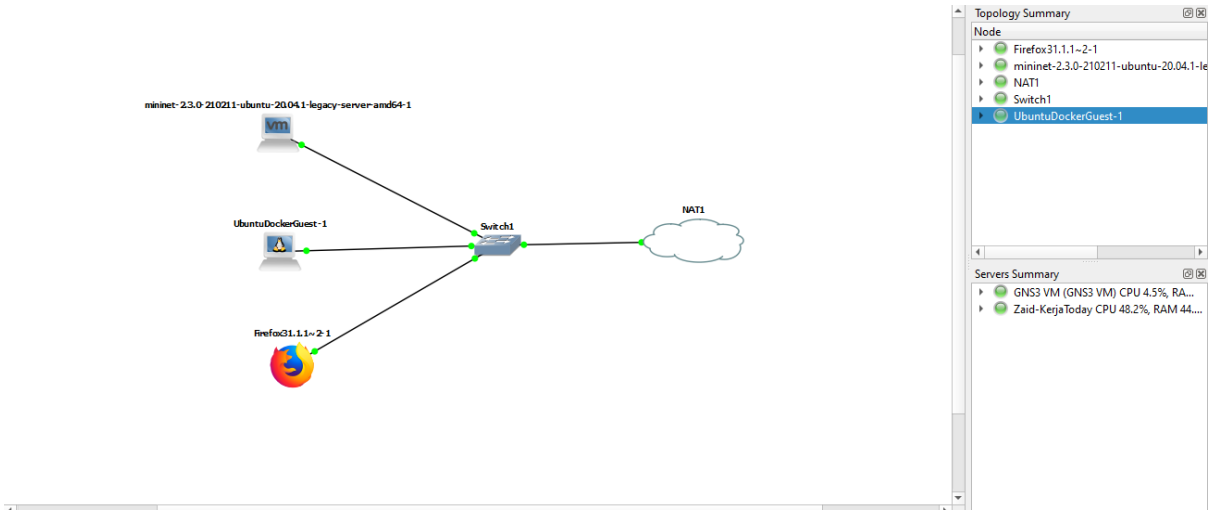

Figure 4.2.1 GNS3 Network Environment

Step 2: Using the Ubuntu Docker as shown in the network environment, install Java and OpenDayLight SDN Controller.

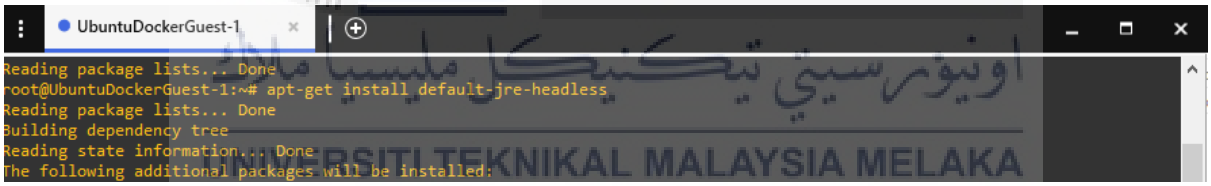

Figure 4.2.2 Installing JAVA

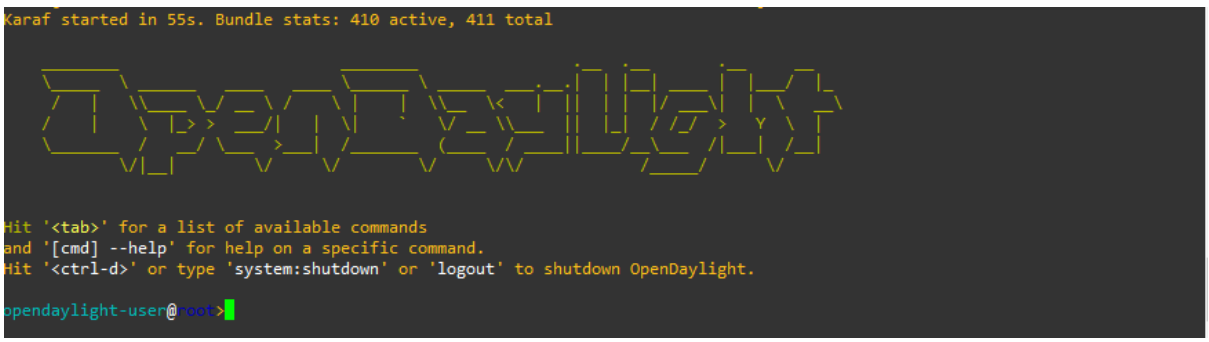

Figure 4.2.3 Starting up OpenDayLight

Step 3: Make sure Ubuntu Docker and Mininet can ping with each other.

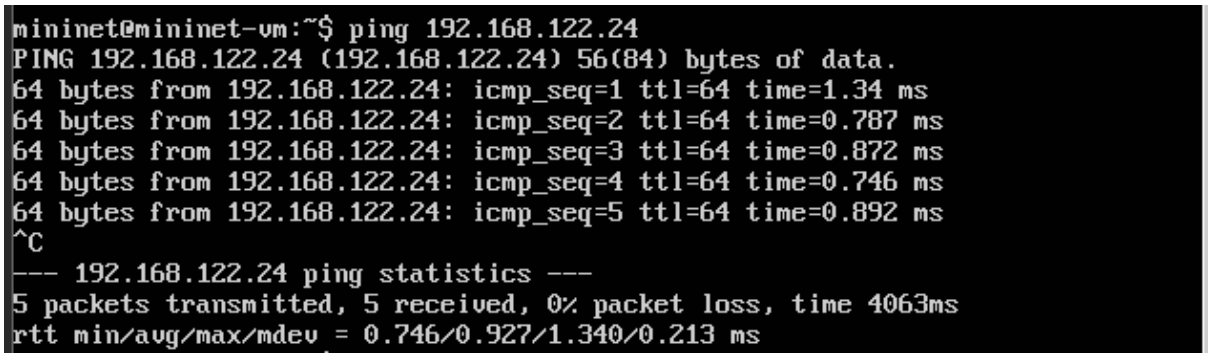

Figure 4.2.4 Pinging Ubuntu Docker

Step 4: On Mininet, create a virtual topology as shown below:

Figure 4.2.5 Create Topology

Step 5: By clicking the topology section, topology that has been created through Mininet is shown.

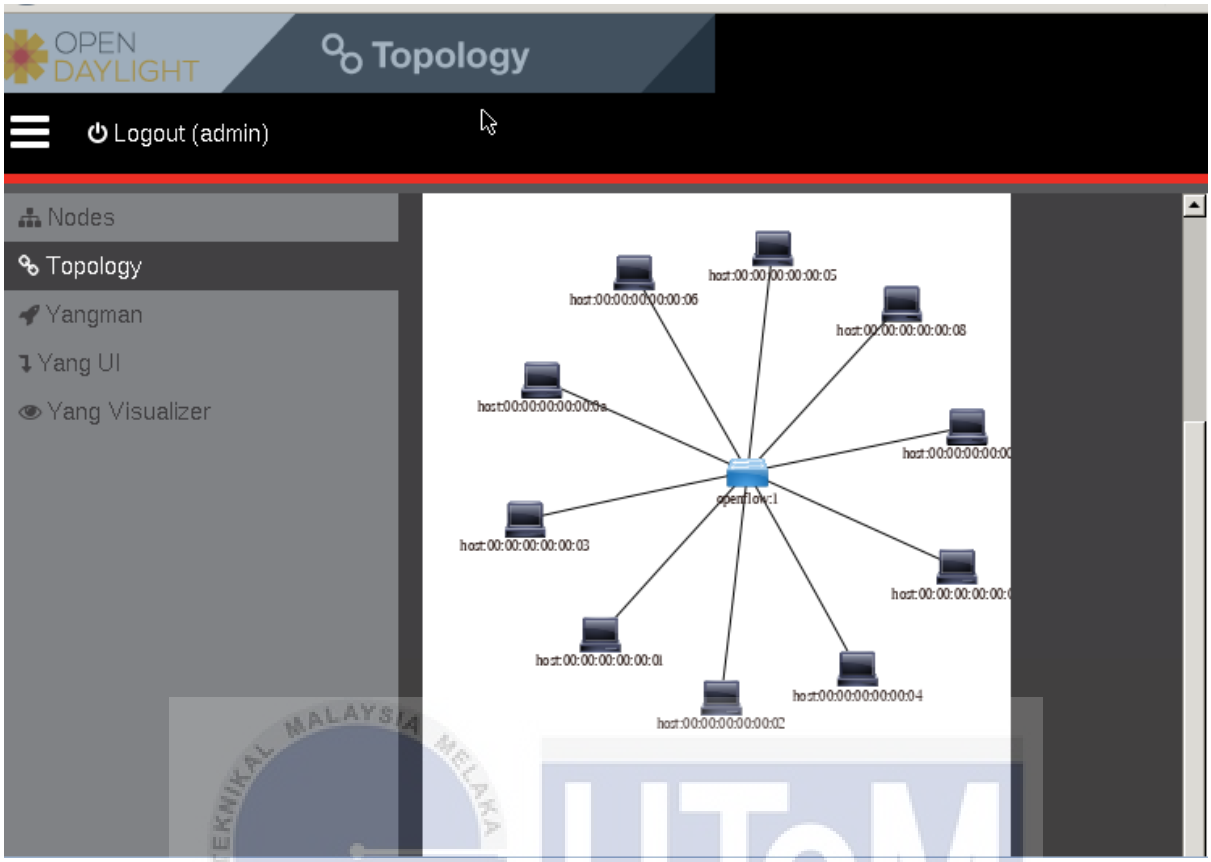

Figure 4.2.6 Topology of created Network on Firefox

Step 6: To check the connectivity between the hosts in the topology using Mininet is as shown below.

```
mininet> h1 ping h9
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
64 bytes from 10.0.0.9: icmp_seq=1 ttl=64 time=0.454 ms
64 bytes from 10.0.0.9: icmp_seq=2 ttl=64 time=0.078 ms
64 bytes from 10.0.0.9: icmp_seq=3 ttl=64 time=0.081 ms
64 bytes from 10.0.0.9: icmp_seq=4 ttl=64 time=0.082 ms
64 bytes from 10.0.0.9: icmp_seq=5 ttl=64 time=0.078 ms
64 bytes from 10.0.0.9: icmp_seq=6 ttl=64 time=0.065 ms
^C
--- 10.0.0.9 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5112ms
rtt min/avg/max/mdev = 0.065/0.139/0.454/0.140 ms
mininet>
```

Figure 4.2.7 Pinging Host 1 and Host 9

4.3 Possible Scenarios

There are three network scenarios in this project:Topology A, B, and C. All of these designs will be created using Mininet, a simulation software that will be used for this project. This project will be completed entirely in a simulation environment. Every network design

for this project includes an SDN controller, switches, and a variety of client connections, all of which must be tested.

4.3.1 Scenario A Single Topology.

This is the first architecture to use a single OpenFlow switch to connect four hosts. Mininet configuration was used to generate this topology. It's referred to as single topology. A single topology consists of a single switch that is connected to several hosts. A single switch with ten hosts is created in this example. This topology was referred to in a Journal with a title Design and Performance Analysis of OpenFlow Enabled Network Topologies using Mininet by the author Idris Zoher Bholebawa and Upena D.Dalal.

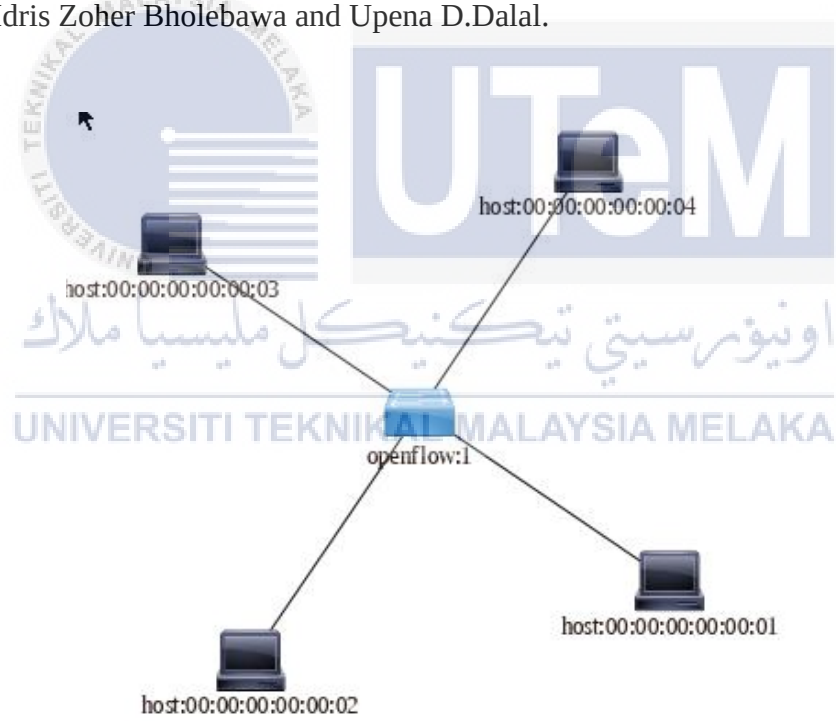

Figure 4.3.1 Topology A

4.3.2 Scenario B Linear Topology

The second design is known as Linear Topology, and it consists of four OpenFlow switches, each of which is connected to a single host. A linear topology is made up of back-to-back switches with a single host (PC) linked to each switch.

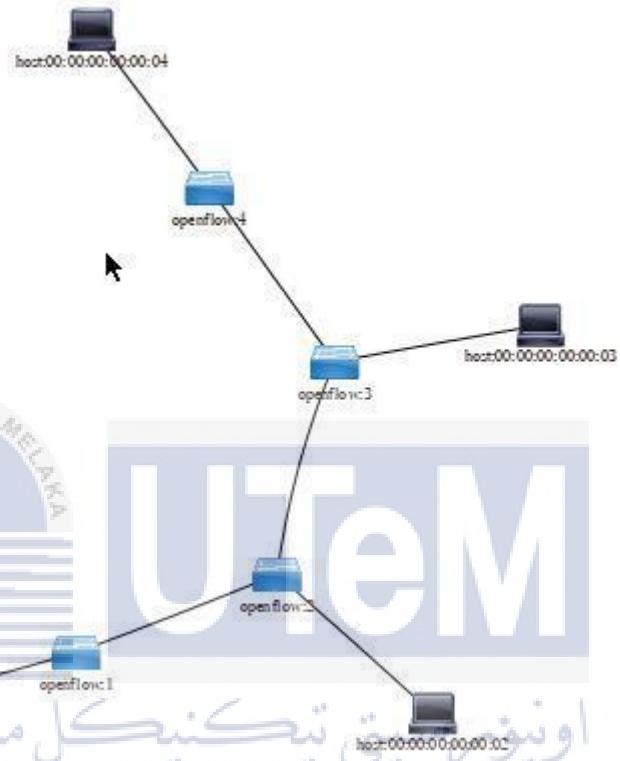

Figure 4.3.2 Topology B

4.3.3 Scenario B Tree Topology

Tree Topology is the third network design, which consists of a core switch and two fanout switches. As a result, two switches are connected to the main switch. Because there is a depth of two, the depth is one two. The fanout command will determine the number of hosts connected to each leaf switch. As a result, there are no hosts connected to the core switch. Hosts are connected to the leaf switches.

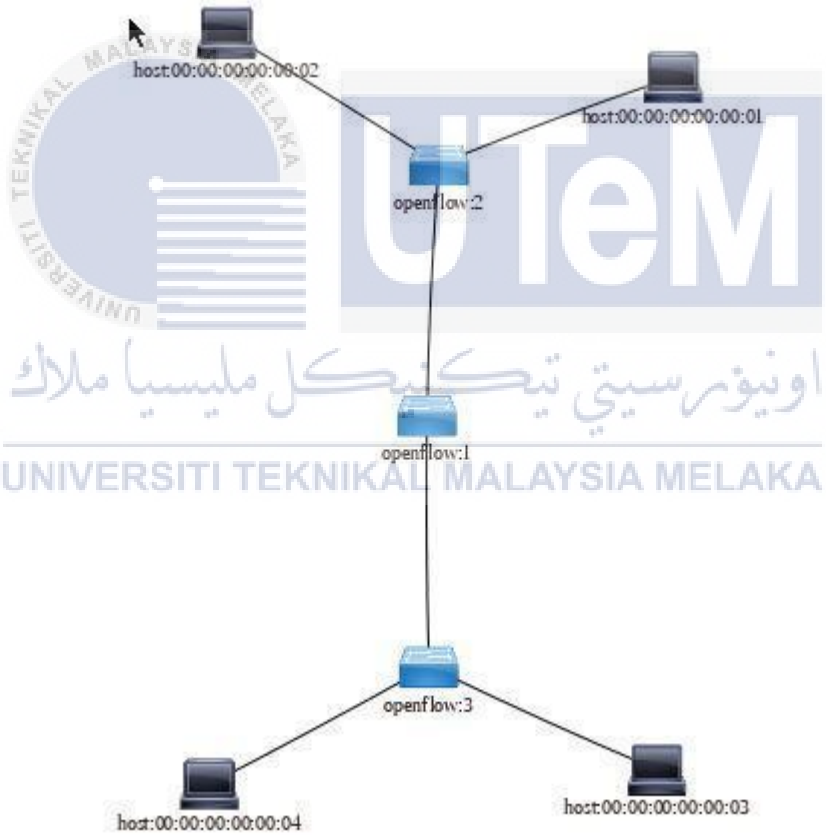

Figure 4.3.3 Topology C

4.4 Metric units of Measurement

We compare load balancing algorithms using attributes like throughput, latency, and jitter in order to test them. The throughput, latency and jitter can be determined mathematically as follows:

Throughput	Number of bits/s
Latency	Time of 2nd packet - time of first packet

Table 4.1 Calculation Formula

4.5 Conclusion

To summarise, this chapter focuses on the creation of a network scenario for this project, which is a key phase in the project's planning. This chapter contains all of the critical design requirements that must be implemented and tested in the following chapter. It also discusses how the architecture for this projected project was designed.

CHAPTER 5: IMPLEMENTATION

5.1 Introduction

This chapter describes the execution of the Load Balancing Mechanisms Analysis project for better network performance using open flow. In this phase, it focuses on the implementation of a number of configurations for data acquired in this project. Setup of the Network scenario, setup and network monitoring and display how to install and manage software. Everything about how this chapter is implemented and configured.

5.2 Environment Setup

5.2.1.Network Scenario Environment Setup

In this section, the method on network configuration was discussed for the project implementation. The implementation steps are shown step by step in this section. Mininet was fully used to simulate and illustrate the details of the project network configuration.

5.2.1.1 Scenario A Topology Environment Setup

The configuration was done using a Mininet VM that was placed on the GNS3 network environment. Mininet VM connected to the switch that also connected to the Ubuntu Docker Guest that holds the OpenDayLight Controller. Through Mininet a topology can be created with a single line. There are several topologies that can be created through Mininet which are Tree, Linear and Single. These three topologies were the most common topology used (Idris Zoher Bholebawa and Upena D.Dalal).

Based on Figure 5.1 below, shows the Topology of Scenario 1. This topology consists of a single OpenFlow switch and 4 hosts connected to it at once.

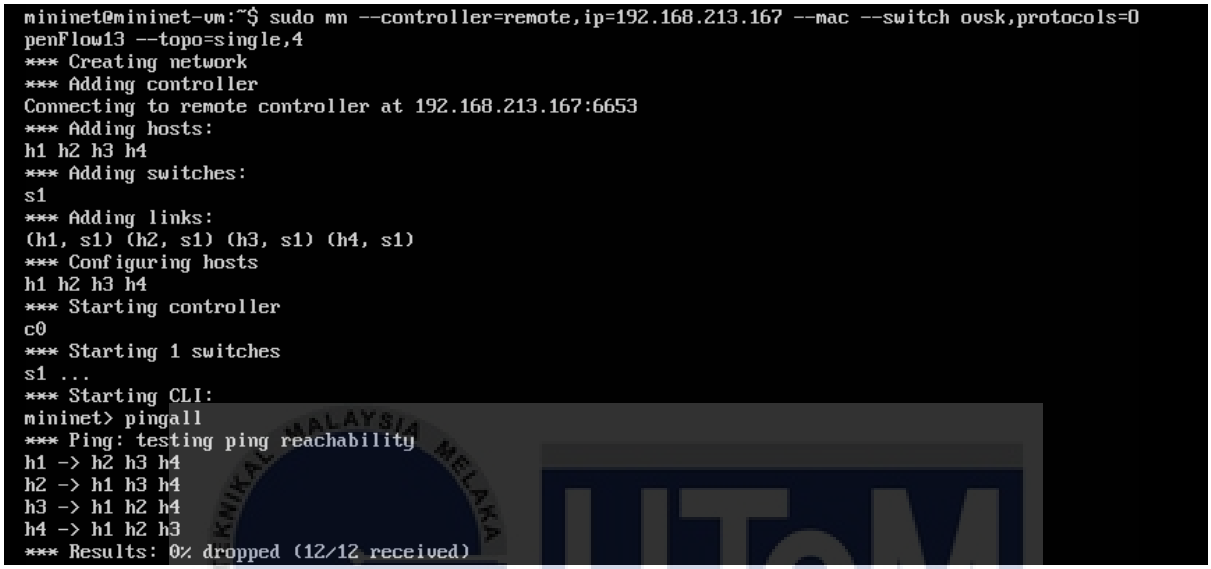

Figure 5.1 Scenario A Topology Configuration

5.2.1.2 Scenario A Topology Environment Setup

The configuration is known as Linear Topology, and it consists of four OpenFlow switches, each of which is connected to a single host. A linear topology is made up of back-to-back switches with a single host (PC) linked to each switch.

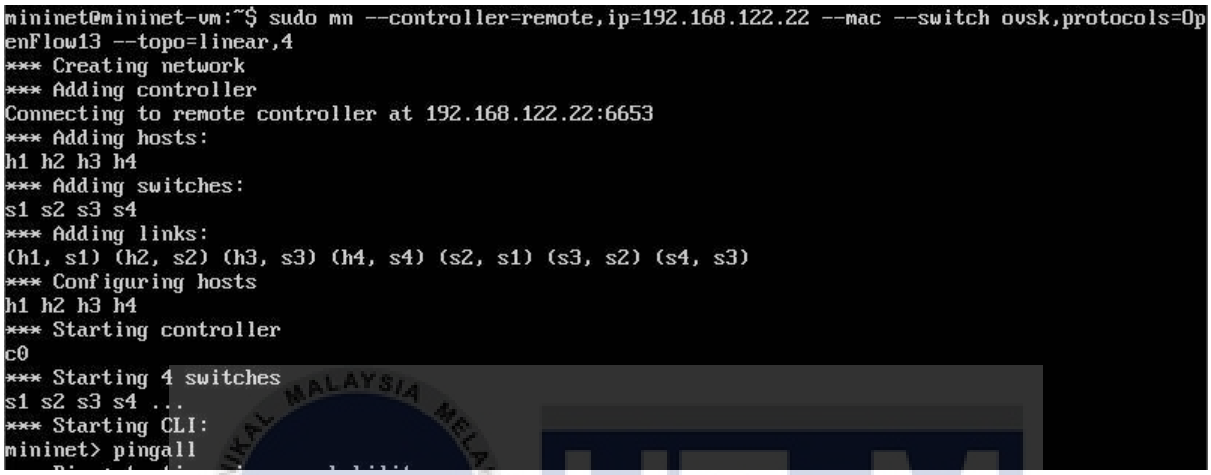

Figure 5.2 Scenario 2 Topology Configuration

5.2.1.3 Scenario C Topology Environment Setup

Tree Topology is the third network configuration on Mininet, which consists of a core switch and two fanout switches. As a result, two switches are connected to the main switch. Because there is a depth of two, the depth is one two. The fanout command will determine the number of hosts connected to each leaf switch. As a result, there are no hosts connected to the core switch. Hosts are connected to the leaf switches.

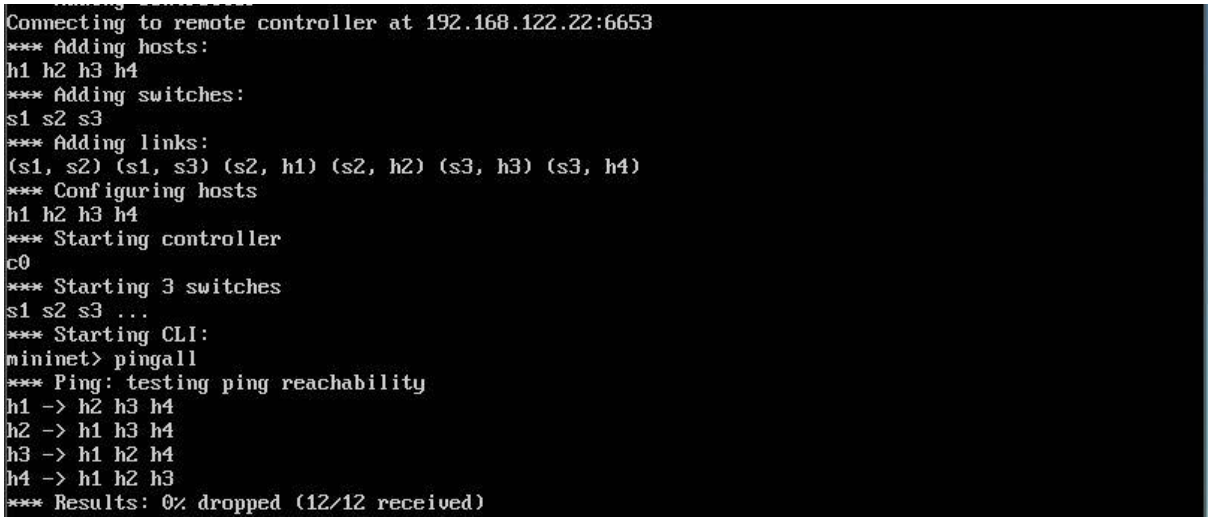

Figure 5.3 Scenario 3 Topology Configuration

5.2.2 Controller Environment Setup

5.2.2.1 Load Balancing Controller Environment Setup

In this project, an OpenFlow controller is linked to a switch. It includes many controller types that may be utilised with Mininet software. The OpenFlow switches were implemented with OpenFlow13 protocols. OpenFlow provides direct access to and modification of network devices' forwarding planes, such as switches and routers, both physical and virtual (hypervisor-based).

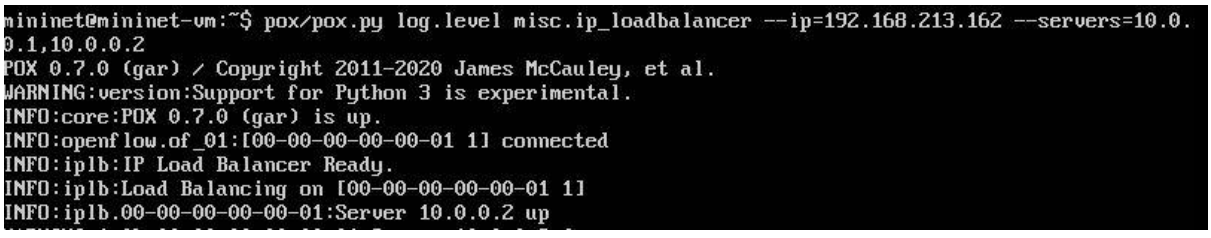

Figure 5.4: Load Balancing configuration on Mininet

5.2.3 Network Monitoring Environment Setup

This project will utilise Wireshark software to test the network and enable the simulation to collect the required parameters. The parameter will be dependent on the results and will be based on throughput, delay, and jitter.

5.2.3.1 Wireshark Environment Setup

Wireshark must be executed as part of the system application to capture packet transfers. The following command is used to start Wireshark as shown in the Figure 5.5

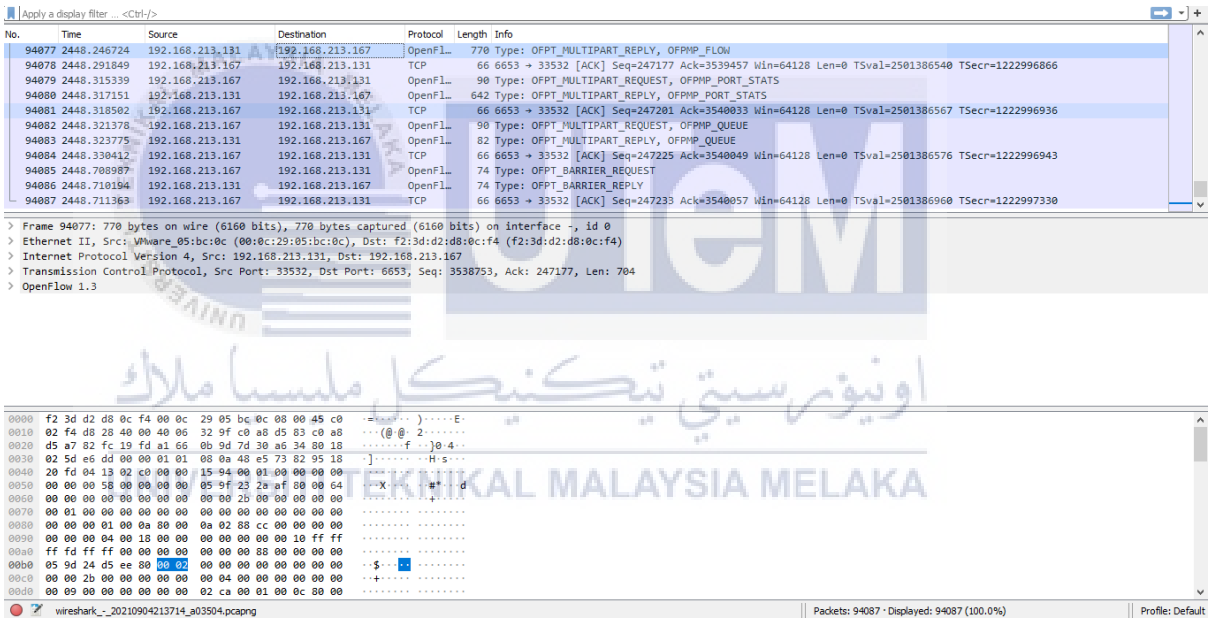

Figure 5.5 Wireshark Environment

5.3 Conclusion

Finally, this chapter discusses how this project will be fully implemented. It mostly focused on how the network and parameters were used, how the technique was implemented,

and how the Testing Phase continued. The following chapter will go through in detail how the data acquired using Wireshark to be tested was captured.

CHAPTER 6: TESTING AND ANALYSIS

6.1 Introduction

The testing and analysis of the proposed solutions will be covered in this chapter. This study employed a data gathering approach to analyse the testing results. Furthermore, this

chapter employed a Mininet experiment setup, as well as a Software Defined Networking controller, to analyse the performance of three types of topologies. This project employs three metric measurements: throughput, delay, and jitter.

This chapter begins with the project's network simulation being implemented. The data will then be gathered utilising a data transmission from the host to the server. In this setting, the Round Robin algorithm is utilised to distribute client requests among a number of servers. All of the data gathered is utilised to evaluate the network's performance.

6.2 Result and Analysis

6.2.1 Bandwidth Utilization

The simulation results for network bandwidth usage are acquired by using the ‘iperf' command in Mininet. As stated in the preceding section, the number of OpenFlow-enabled switches required for the construction of single, linear, and tree topology networks for a common 4-host design, as shown in Table 6.1.

Network Topologies	A Single	B Linear	C Tree
Number of OpenFlow Controllers	1	1	1
Number of OpenFlow-enabled Switch	1	4	2
Number of Host	4	4	4
Maximum Utilized Bandwidth(Gbps)	5.00	4.99	5.37
Minimum Utilized Bandwidth (Gbps)	4.90	4.30	4.90

Table 6.1 Bandwidth Utilization

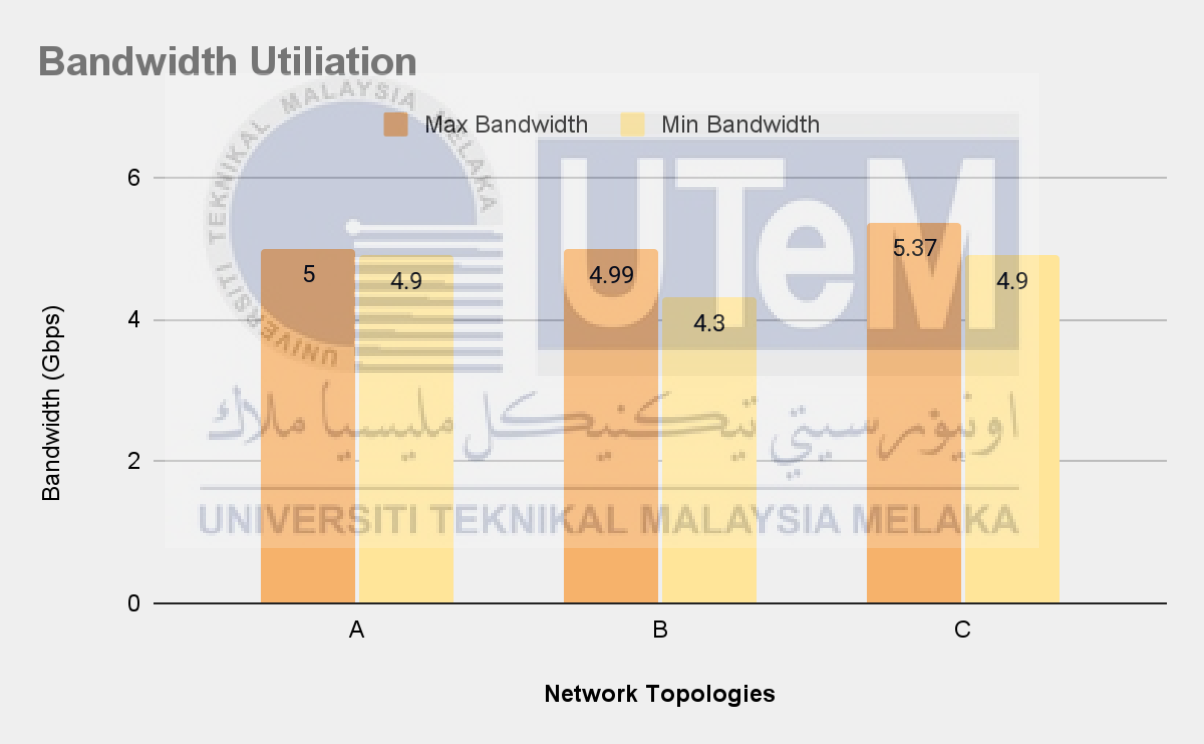

Chart 6.1: Bandwidth Utilization

According to the results, the bandwidth utilized in linear topology is the least and the most in tree topology, as shown in Chart 6.1. In a B linear topology, the connection between a switch and a host is one-to-one; one host is linked to one switch, and the switches are connected to each other for end-node communication. For the number of nodes, the overall bandwidth

usage of the network is restricted. Whereas with C tree topology, the design is dispersed yet centrally regulated, resulting in higher overall bandwidth usage.

6.2.2 Packet Transmission Rate

The three topologies are then compared based on Packet Transmission Rate (PTR). The PTR for each of the three topologies is compared in Table 6.2 and graphed in Chart 6.2.

Number of Packets Transmitted	A	B	C
5	**4077 ms**	**4094ms**	**4097 ms**
10	**9223 ms**	**9209 ms**	**9212 ms**
20	**19437 ms**	**19453 ms**	**19447 ms**
30	**29676 ms**	**29673 ms**	**29695 ms**

Table 6.2 Packets Transmission Rate

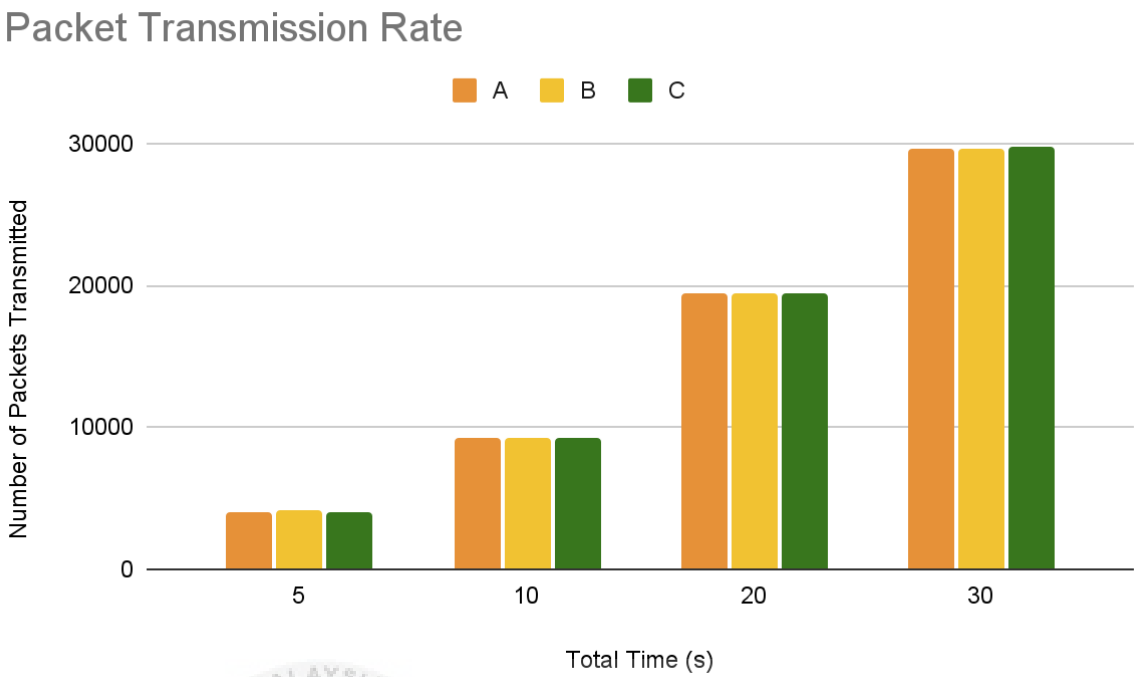

Chart 6.2: Packets Transmission Rate

Based on the above PTR results, the overall time required by the three OpenFlow network topologies for various packet transmissions is nearly the same. Because all nodes in an OpenFlow network behave the same way, the OpenFlow network is active for the same time interval to execute diverse network topologies for a similar packet transmission rate.

6.2.3 Round Trip Time Delay

Following that, a comparison of three topologies is made based on the time it takes for nodes in a network to communicate with one another. This may be accomplished by running a ‘ping' connection test to determine the round-trip time (rtt) between nodes. The least and maximum round-trip delay between nodes for various network topologies with varying PTR is

listed in Table 6.3 and graphically displayed in Charts 6.3 and 6.4 for minimum and maximum delay, respectively.

Number of Packets Transmitted	A		B		C	
	Min.rtt	Max.rtt	Min.rtt	Max.rtt	Min.rtt	Max.rtt
5	0.042ms	0.182ms	0.045 ms	0.063ms	0.046ms	0.160ms
10	0.041ms	0.182ms	0.045ms	0.212ms	0.049ms	0.171ms
20	0.040ms	0.161ms	0.043ms	0.371ms	0.040ms	0.269ms
30	0.041ms	0.254ms	0.042ms	0.277ms	0.029ms	0.251ms

Table 6.3 Round Trip Time Delay

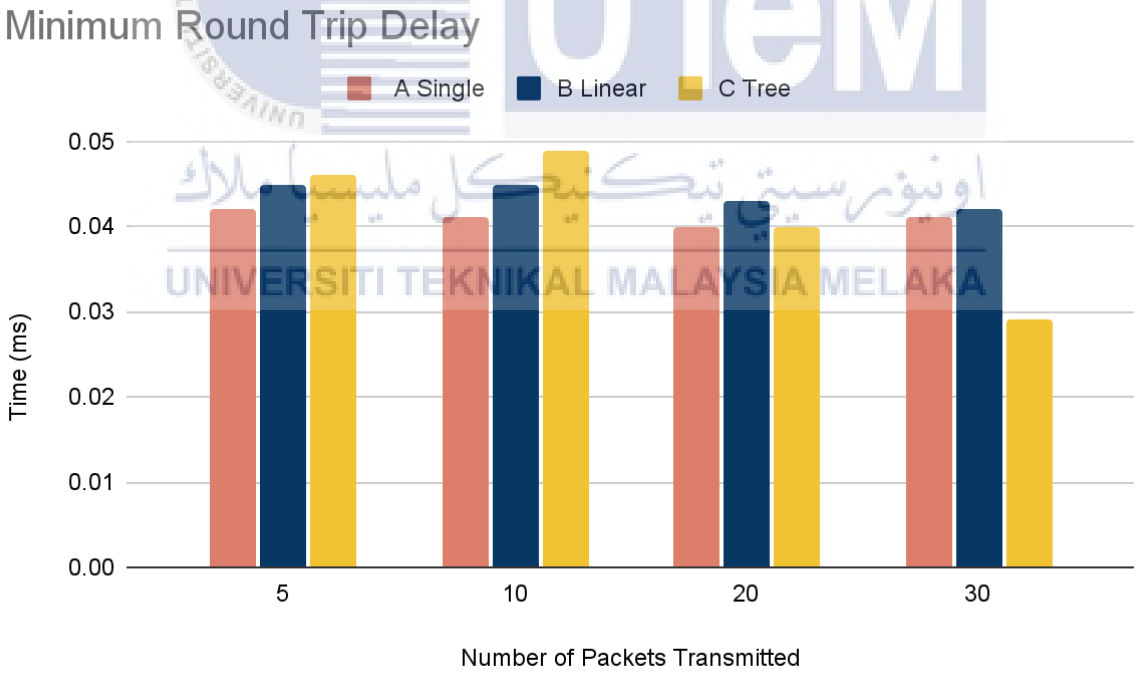

Chart 6.3: Minimum Round Trip Delay

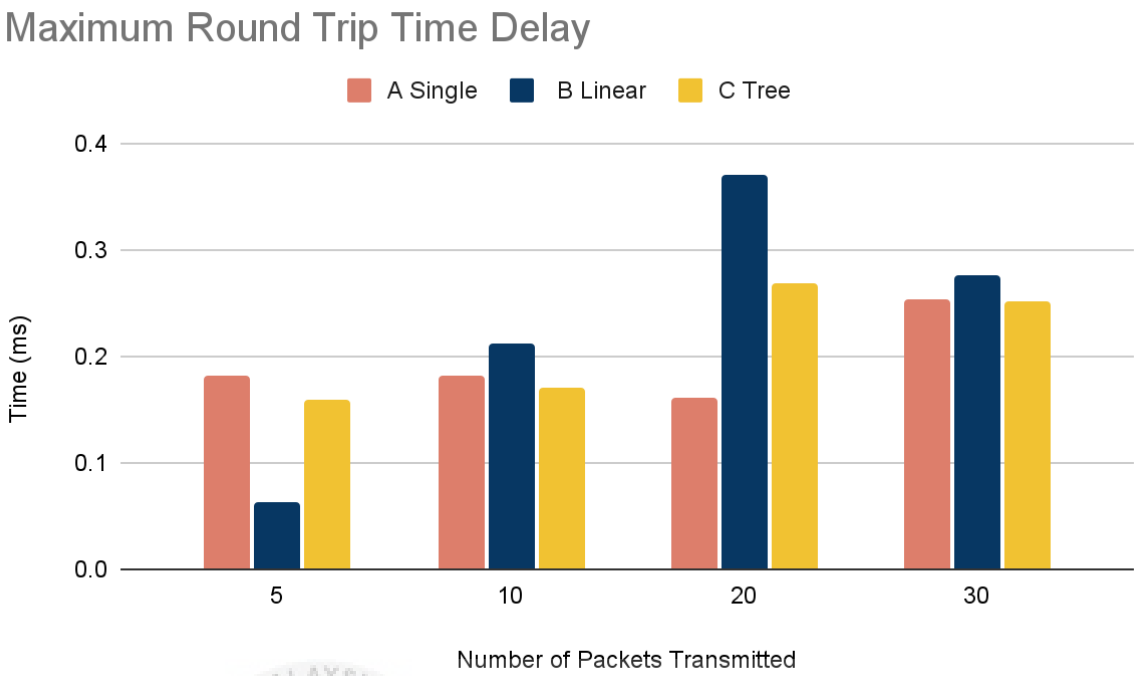

Chart 6.4: Maximum Round Trip Delay

According to the Chart 6.4, a linear topology takes longer to transmit a packet to its target node than a A single or B tree architecture. Because the number of hops between end nodes is increasing, more propagation time is necessary for intermediate nodes to send packets to their precise destination. In contrast, A single topology takes the shortest amount of time to transmit a packet to its destination. Because all nodes are linked by a single OpenFlow-enabled switch thus. A single topology, packet delivery will be quicker.

6.2.4 Throughput

Finally, throughput analysis of a network is used to compare fundamental OpenFlow topologies. The amount of data sent from source to destination in a particular time period is described as network throughput.

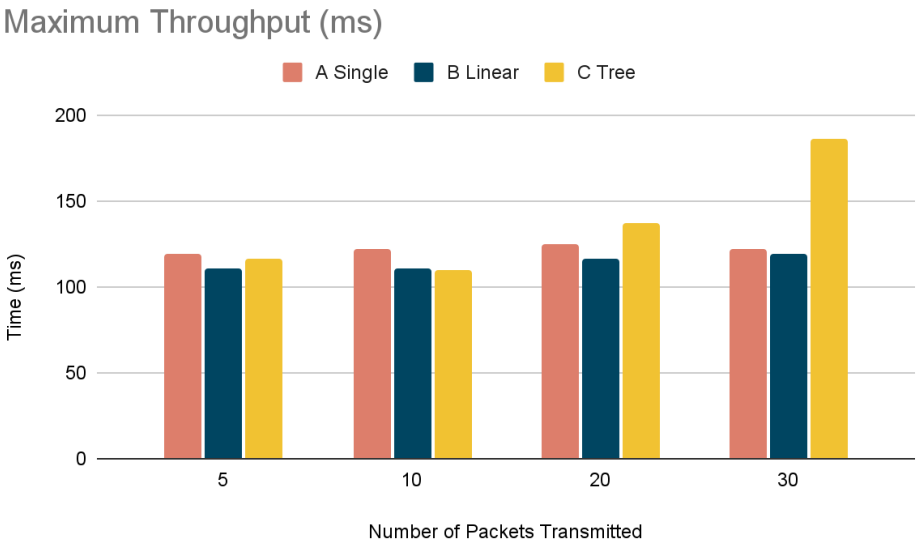

Chart 6.5: Maximum Throughput

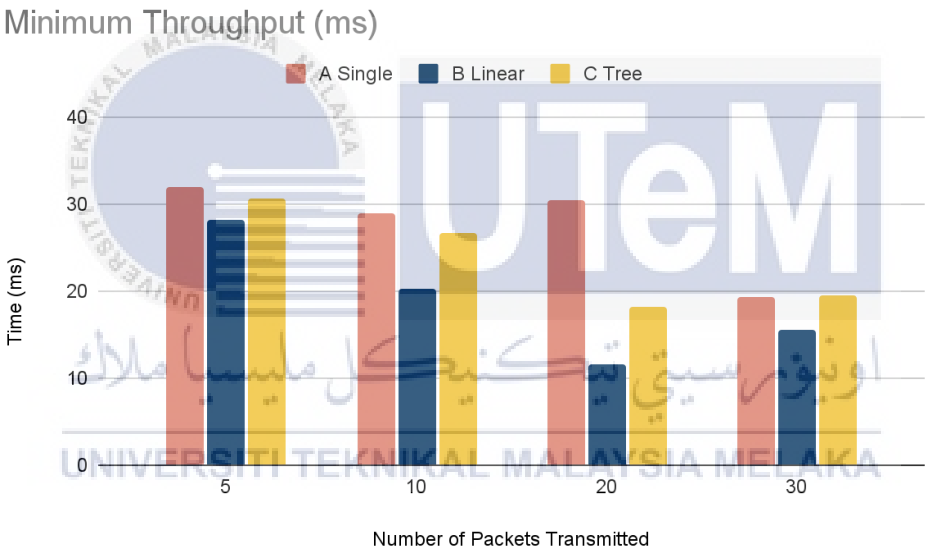

Chart 6.6: Minimum Throughput

Charts 6.5 and 6.6 illustrate a minimum and maximum throughput graph in relation to packet transmission rate. According to the produced graph, the throughput of the B linear topology is very low when compared to the other two topologies. This is because the bandwidth utilisation is low and the overall round-trip propagation latency between nodes is high.

Meanwhile the A single topology network has the highest throughput when compared to the other two topologies, this is due to the single topology having only one switch and only two hops between any two nodes in a network. Thus, in a single topology, the delay is less and the throughput is greater, but in a linear topology, the delay is more and the throughput is lower.

6.2.5 Summary Table

Parameters	A Single	B Linear	C Tree
Max Bandwidth Utilization (Gbps)	5	4.99	5.37
30 Packets Transmitted Rate (ms)	29676	29673	29695
Maximum Round Trip Time Delay (ms)	0.254	0.277	0.251
Maximum Throughput (ms)	121.95	118.81	185.17

Table 6.4 Summary Table

6.3 Conclusion

In summary, this chapter discusses how the project was tested, as well as how the data was analysed and presented in the graph view. Based on the performance study of the suggested network topologies and the discussion of the results, we can infer that the A single topology outperforms the other two B linear and C tree topologies, with certain restrictions.

Meanwhile, when compared to the other two topologies, the A single topology network has the best throughput. This is owing to the A single topology having just one switch and only two hops between any two nodes in a network. As a result, in a single topology, the delay is less and the throughput is higher, but in a B linear topology, the delay is larger and the throughput is lower.

A Tree topology may be easily implemented with a single command. The complexity of a network is somewhat more than that of a single and linear topology, but the number of hops between hosts is the same. And, in terms of speed, the performance is increased when compared to linear topology but restricted when compared to single topology.

The total performance of a single topology network will undoubtedly decrease as the number of hosts rises, but this is not the case with B tree topology because the load is spread. In addition, the installation space and cost configuration are less than with linear topology.

CHAPTER 7: PROJECT CONCLUSION

7.1 Introduction

The purpose of this chapter is to provide a summary of the research conducted in this project. First, this chapter will look at whether the project's goal has been met. Second, we discuss the importance of the study's details. Furthermore, ideas for future study paths are emphasised, as well as how this project will be enhanced in the future.

7.2 Project Summarization

The primary goal of this project is to simulate a network scenario of load balancing utilising upcoming technologies such as OpenFlow and tools such as GNS, Mininet and OpenDaylight Controller for the improvement of network performance and reduce the cost of inventing a network environment.. With the integration of Mininet into the Software Define Network Controller project using GNS3, a three-network scenario may be implemented using a virtual network, which is a simulation environment, without the use of expensive network hardware.

The second goal is to create dynamic load balancing utilising various approaches in order to get better outcomes and better performance. Using the pox controller and load balancing available in OpenFlow, this project implemented a controller and a basic round robin load balancing algorithm to be used and to analyse how the server will be controlled by the load balancing controller can result in a better result and higher performance, or vice versa.

Finally, the Round Robin method must be configured in the load balancer. Round robin is the algorithm used in this project. The round robin algorithm, which is the algorithm, assigns equal amounts to each process in a circular sequence, managing all processes without priority. It is clear from the output that the controller instructs the packet to send to the nearest accessible server.

7.3 Project Contribution

The contribution mentioned in the first chapter includes. First, a new network scenario design with and without load balancing for improved data transmission. In this project, three network scenarios are chosen and implemented using a load balancing controller for analysis in Chapter 6. Each scenario utilised a different host connection to see if the results were different.

The second step is to implement the Round Robin method at the network controller. A round robin implementation is utilised for the controller. Several tests were run utilising network simulation in three different settings for analysis. Several indicators are used to assess the effectiveness of the load balance system. This project concentrated on throughput, latency, and jitter.

The third is the Round Robin Load Balancing. In addition, Wireshark, a free and open-source packet analyzer, is used in this project. Software and communications protocols are developed as well as network maintenance, analysis, and teaching. OpenFlow was also used in

this project. A flexible network protocol, OpenFlow manages and directs traffic across routers and switches from different manufacturers. As a result, routers and switches may be programmed independently of their physical components

7.4 Project Limitation

According to the study's primary focus on load-balancing mechanisms that reduce throughput, latency, and jitter for better network performance, this project constraint exists. A module for traffic categorization has been created. However, the accuracy, precision, and memory of traffic classification were not addressed in this study. To make matters worse, each controller utilises a different programming language, such as Python scripts. The latest OpenDayLight Controller uses JDK 11 and it can be used in the Ubuntu Docker Container as it is only used up to JDK 8. Hence, the old OpenDayLight Controllers were used. The study of SDN in GNS3 is still not widely used thus making this project hard.

7.5 Future Works

This research was focused on how load balancing affects the network performance in reducing throughput, delay and jitter. However the scalability of the load balancer is not discussed. In the future the test on load balancing scalability can be tested using an SDN controller. Other upcoming projects involve offering alternative traffic classification approaches that may be analysed, as well as testing Transaction Rate and Response Time as metric measurements. Other controllers can be used instead of OpenDayLight controllers to obtain better results. As a result, in the event of a failure.

7.6 Conclusion

This chapter provides a summary of the research conducted in this study. First, go through the research goals. Second, project contributions are no longer utilised. Finally, recommendations for future study directions are emphasised.

REFERENCES

Mishra, S and AlShehri, M.A.R (2017). “Software Defined Networking: Research Issues,Challenges and Opportunities” Indian Journal of Science and Technology. 10(29). pp 1-9

Rishabh, K, Angadi, K, Chegu, K, Harikrishna, D.H and Ramya, S (2017). "Analysis of Load Balancing Algorithms in Software Defined Networking,"2nd International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS). Bangalore, 2017. pp. 1-4.

Yu, J,Wang, Y,Pei, K,Zhang, S and Li, J (2016). "A load balancing mechanism for multiple SDN controllers based on load informing strategy,"18th Asia-Pacific Network Operations and Management Symposium (APNOMS), Kanazawa, 2016, pp. 1-4.

Craig, A, Nandy, B, Lambadaris, I and Smith, P.A (2016). "Load balancing for multicast traffic in SDN using real-time link cost modification," 2015 IEEE International Conference on Communications (ICC), London, pp. 5789-5795.

Hikichi, K, Soumiya, T and Yamada, A (2016). "Dynamic application load balancing in distributed SDN controller,"18th Asia-Pacific Network Operations and Management Symposium (APNOMS), Kanazawa. pp. 1-6

Joshi, N and Gupta D (2019). “A Comparative Study on Load Balancing Algorithms in Software Defined Networking.”Ubiquitous Communications and Network Computing (pp.142-150)

Sufiev, H and Haddad, Y (2016). "A dynamic load balancing architecture for SDN,"IEEE International Conference on the Science of Electrical Engineering (ICSEE). pp. 1-3.

Yang, X and Wang, L (2018). “SDN Load Balancing Method based on K-Dijkstra.”International Journal of Performability Engineering. 14. pp. 709-716

Lara, A., Kolasani, A., & Ramamurthy, B. (2014). Network innovation using OpenFlow: A survey. IEEE Communications Surveys and Tutorials, 16(1), 493–512.

Open Networking Foundation (ONF). (April, 2012). Software defined networking: The new norm for networks. White Paper. From https://www.opennetworking.org/images/stories/downloads/openflow /wp-sdn-newnorm.pdf

Naous, J., Erickson, D., Covington, G. A., Appenzaller, G., & McKeown, N. (2008). Implementing an OpenFlow switch on the NetFPGA platform. Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communication Systems (pp. 1–9). CA, USA.

Shimonishi, H., Takamiya, Y., Chiba, Y., Sugyo, K., Hatano, Y., Sonoda, K., Suzuki, K., Kotani, D., & Akiyoshi, I. (2012). Programmable network using OpenFlow for network researches and experiments. Proceedings of the Sixth International Conference on Mobile Computing and Ubiquitous Networking (pp. 164-171). Okinawa, Japan.