Design and implementation of SDN fault tolerant framework and security services on-demand

M Mbodila

orcid.org 0000-0003-4158-9037

Thesis accepted in fulfilment of the requirements for the degree *Doctor of Philosophy in Computer and Information Sciences with Computer Science and Information Systems* at the North-West University

Promoter: Prof B Isong

Graduation: July 2024

Declaration

I, Munienge Mbodila, declare that this thesis, titled "**Design and implementation of SDN controller fault tolerance framework and security services on-demand**" is my work completed at the North-West University, Mafikeng Campus, and has not been submitted in any manner for a degree to any other university or institution of higher learning before being published. All informational sources utilised in the text and references have all been properly cited and have never been submitted for any university degree award.

2024 / 04 / 09

Signature ———————————— Date ————————————

Munienge Mbodila

Approval

Bisong 8th May 2024

Signature ———————————— Date ————————————

Supervisor**:** **Prof Bassey Isong**

Department of Computer Science
Faculty of Agriculture, Science, and Technology
North-West University,
Mafikeng Campus
South Africa

Supervisor**:** **Prof Naison Gasela**

Department of Computer Science
Faculty of Agriculture, Science, and Technology
North-West University,
Mafikeng Campus
South Africa

Dedication

This research thesis is dedicated to my children, family, as well as friends who have supported me during this journey.

Acknowledgements

I want to thank and honour Almighty God for enabling me to complete this challenging journey. I give God praise for providing my life, strength, knowledge, education, discernment, and provision, and for guiding me to the successful completion of my Ph.D. Above all, He gave me the courage and hope when I needed it most. My supervisor, Prof Bassey Isong, deserves my gratitude for his perseverance, support, suggestions, counsel, and constructive criticism while I worked on this research endeavour. It was not an easy journey, and I appreciate his encouragement for me to pursue this Ph.D.

A special thank you goes out to my co-supervisor, Prof Naison Gasela, for his assistance and participation in this project. I also appreciate the assistance and financial support provided by the North-West University, Mafikeng Campus for this programme.

In addition, I want to thank Walter Sisulu University, where I work, for all the financial support they have given me during this journey. To all my family, friends, and colleagues from work to my elder brother Dr Lazare Mabanza (PhD), who set a good example for me, Thete Masonga and her family, Niclette Nsakwa and her family, and Sarah Mabanza, thank you for your support and encouragement.

I want to also thank Francine Kikunga, Fhulufhedzani Mavhungu, and the kids for being there for me during this journey, as well as Dr Mpiti my rector, and Dr Bwowe, my dean, for their encouragement and motivation toward this PhD. To Prof Clever Ndebele, Prof Verona Leendertz, Miss Ntara Mdoda, Mavuso Anele, Esan Bayo, Femi Elegbeleye, Sarah Mbi, and Nosipho Dladlu for your support, I appreciate you all.

To my late brothers Zuka and Diona; and my father Godet Mabanza, this work is also for you; may your souls rest in eternal peace.

For my mother, Mrs Mabanza Mikwimi Yvonne, thank you for your support, and faith; I am incredibly appreciative to God for having you.

Finally, I want to thank and appreciate my wife, Mrs Blandine Muhandji Mbodila for all her love, support, and inspiration, and thanks to Preddie Mbodila, Graddi Mbodila, and Merdi Mbodila for motivating me to make it this far despite their requests, demands attention and interruptions. Without you guys, this was never going to mean anything. You are such a blessing to me, and I love you.

Last but not least, I would like to thank myself, for being strong even when I was down, and for self-motivation and perseverance toward this PhD, I did not give up even when I had reasons to give up.

Abstract

Software-defined networking (SDN) revolutionizes network management by separating control and data planes, improving programmability, and automating operations. However, the centralized SDN control plane which houses the network intelligence presents challenges such as single points of failure with single controllers or added complexity with multi-controllers. Despite these challenges, SDN offers critical functionalities for security and on-demand services. This thesis addresses SDN's challenges by focusing on fault tolerance and security management. The study proposes a fault-tolerance model (FTM) to improve SDN reliability and applies it to security management. The FTM prioritizes fault monitoring, detection, and recovery as well as ensuring scalability and performance across networks of varying sizes: small to large-scale networks. By utilizing design science research methodology, this study designed an FTM which employs backup controllers and stationary agents for small networks, while introducing a novel controller placement technique for large networks. In addition, an on-demand security service framework that leverages SDN functionalities to improve network protection is designed. Performance evaluation in Mininet simulations based on OpenFlow and Ryu SDN controllers using different evaluation metrics demonstrates the effectiveness of the proposed FTM, with promising performance in certain scenarios. Particularly, the OpenFlow controllers exhibit lower latency rates after failure in small-scale networks, while Ryu controllers outperform in medium to large-scale networks in terms of the number of controllers deployed and network availability. These findings provide customized implementation strategies for different network sizes and recommend the adoption of the frameworks in real-world SDN environments to enhance network resiliency and dependability.

Keywords: *Software-defined networking, fault tolerance, OpenFlow, Ryu, controller placement, security services, on-demand*

Table of contents

Contents

List of Figures

List of Tables

Definition of Concepts

Authentication is the procedure for identifying and having an identity confirmed by providing a username and password to access a computer system or a network [1].

Centralised network: A network architecture where users can only communicate with one another through a central authority. Since communication between all parties involves using a single centralised source, its disappearance would make communication impossible [2].

Fault: An incident that prevents the system or the network from operating properly and lowers its performance [3].

Failure: is an event or circumstance where something does not perform as expected or desired. This is one of the greatest threats in a system or network that directly affects how services are delivered [3].

Fault detection: It is a process of finding a hardware or software fault [4].

Fault management is focused on identifying, isolating, and fixing a fault in a network [5].

Fault tolerance management is part of the fault tolerance framework that deals with fault detection, fault isolation, fault recovery, and fault fixing in a network [5].

Fault tolerance: This refers to a system's capacity to maintain uninterrupted operation if one or more of its components fail [5].

IP address: Also known as an internet protocol address, it is a number that identifies a machine or any device on a network [6].

Medium/scale network: A type of network that uses a large geographical area and connects more devices in the form of a wide area network [7].

Network: A collection of linked computers that can exchange information [8].

Network downtime refers to the inability to access all or a portion of a network due to a connection, software, or hardware issue [8].

Network Failure: A network failure refers to a situation where a computer network or a part of it ceases to function correctly, leading to disruptions in communication or service delivery [8].

Network node: It is a device connected to the network such as a PC, server, router, controller, etc [3].

Network monitoring: Network management is the process of routinely inspecting a computer network for issues or defects [9].

Network partition: This is the concept known as network segmentation, which involves the division of a network into portions or domains [6].

Network service: A network service facilitates network activities, usually open system interconnection model services as stipulated in the network protocols of the application layer [10].

Network slicing is a process for building numerous different virtualised and logical networks over a single multidomain infrastructure [1].

Network function virtualisation: It is a new technology that substitutes sophisticated network appliances with commodity server hardware to provide more flexible, effective, and scalable services [1].

Network state synchronisation: This is a procedure that involves controlling the execution of several threats to assure the intended result without tampering with shared data and avoiding any deadlocks or race circumstances [9].

Network reliability: This is the quality of any computer network component that consistently functions following its requirements, whether it is software, hardware, or a network [11].

Network topology: This represents the logical and physical connection between network nodes, the network's interconnections, and nodes' schematic organisation, or a combination of these [12].

Network traffic: This is the percentage of uptime in a network system over a given period, and is known as network availability [3].

Network recovery: On a computer network, this is the procedure for recovering and returning to regular working activities of a network [4].

Network virtualisation: When network hardware and software resources are merged with network operations into a single, software-based administration entity, the results are referred to as a virtual network [12].

Replication: This is a technique of copying data also known as replicating data from one point to another in a network [11].

Security service: A competence that helps achieve one or more security objectives. Access management: authentication, and key management are a few examples of security services [13].

SDN controller: This is a program that controls flow for better network management and application performance in a software-defined networking (SDN) architecture [3].

Switch: It handles traffic forwarding and control functions, enabling networked devices to successfully communicate with one another [14].

Small-scale network: A type of network that has few machines connected to it, also known as a local network area [14].

Stationary agent: This is a piece of software that keeps an eye on and controls communication and traffic moving through the network device it is connected to [14].

Synchronisation: This is the precise coordination of multiple events or mechanical devices [9].

Virtualisation: This allows for the creation of a virtual copy of a variety of things, including hardware platforms, operating systems, storage devices, and network resources [13].

Virtual networking: This is when network resources are separated from the underlying physical network infrastructure in virtual networking [10].

List of acronyms

AHP: Analytic Hierarchy Process

APA: Application Plane

API's: Application Plane Interfaces

CPI's: Control Plane Interface

CP: Control Plane

CPP: Controller Placement Problem

CCPP: Capacitated Controller Placement Problem

CNPA: Cluster-Based Network Partition Algorithm

CE: Customer Edge

CLI: Command Line Interface

Cnt_ID: Controller ID

Cont_State: Controller State

Cnt_loctn: Controller Location

DOT: Distributed OpenFlow Testbed

DBCP: Density-Based Controller Placement

DCP: Dynamic Controller Provisioning

DCPP: Dynamic Controller Placement Provisioning

DC: Data Centre

FP: Forwarding Plane

FTF: Fault Tolerance Framework

FMS: Fault Management System

HPP: Hypervisor Placement Problem

ID: Identification

IT: Information Technology

IaaS: Infrastructure as a Service

IS: Information Systems

IP Address: Internet Protocol Address

IP: Internet Protocol

MLog: Message Loge

NaaS: Network as a Service

NP: Nondeterministic-Hard Problem

NMP: N-modular Programming

NOX:	A piece of the software-defined networking (SDN) ecosystem
NWU:	North-West University
OS:	Operating Systems
PaaS:	Platform as a Service
POX:	Python-Only Version of NOX
QoS:	Quality of Services
Rsp_Time:	Response Time
SaaS:	Software as a Service
SDN:	Software-defined Networking
SDN-FTF:	Software-defined Networking Fault Tolerance Framework
SLAs:	Services-level Agreements
SA:	Stationary Agent
sDB:	Stable Database
TCP/IP:	Transfer Control Protocol / Internet Protocol
UCPP:	Uncapacitated Controller Placement Problem
VSDN:	Virtualised Software Defined Networking
VCPP:	Virtualised Capacitated Controller Placement Problem
WAN:	Worldwide Network

Chapter 1

Introduction and Background

1.1. Outline

This chapter provides an introduction and context to the research field. The goals, motivations, contributions, and organization of the thesis, as well as its scope, are discussed in detail in this chapter. This comprehensive overview sets the stage for the subsequent chapters and provides a clear understanding of the research's objectives and significance. It also outlines the structure of the thesis, guiding the reader through the various sections and their interconnections. This chapter serves as a roadmap for the entire thesis, highlighting its key aspects and their relevance to the broader research field.

1.2 Introduction

The Internet has witnessed significant innovation due to the evolution of network architecture over the years. However, operating the network itself remains a considerable challenge. In a traditional network, the complex control plane that resides atop all switches and routers is the primary source of this issue. Numerous network manufacturers produce these network devices, and the data plane is designed to be governed by proprietary protocols. Given that the control plane was an integral part of each network device, distributed optimization of network control was inherently challenging. This rapid transformation in the actual network landscape is characterized as smarter, faster, more reliable, and more scalable to adapt to the dynamic behaviour of modern networks in terms of bandwidth and computing capability. Virtualization, data storage, and cloud-based services have become challenging for network operators, cloud service providers, or users in the IT industry [15]. These challenges are posing and raising numerous concerns in maintaining network traffic, access control, and software verification for on-demand security services [15] [16]. As a result, network administrators face significant difficulties regarding fault management and monitoring.

A new networking paradigm, Software-Defined Networking (SDN), was developed to enable the next generation of networks to become programmable, offering a variety of manageable, flexible, and affordable choices [17]. This characteristic makes it ideal for addressing numerous limitations imposed by high network traffic in the revolutionary networking paradigm known

as SDN. In SDN, network devices are manageable and make forwarding decisions using simplified control through software. To communicate in SDN architectures, network devices use the SDN controller, which provides them with suitable forwarding decisions by using different protocols to establish a secure connection during traffic flow on the network. The controller also referred to as a special node [5] [15], is the essential element of SDN architecture. It operates the standard cloud-based on-demand services platforms as a network operator. The programmability of the SDN controller allows it to coordinate and manage network behaviour by assigning infrastructure, sharing applications, and granting network access to end-user devices and services in a cloud-based environment. For effective communication between the network devices and the SDN controller, different types of protocols can be used, but the most important one is OpenFlow [15]. This protocol describes the control messages that grant the SDN controller authorization to connect to network nodes' security, read their current states, and configure forwarding instructions [2][17]. The controller is the central component of the SDN architecture, and in cases where there is a fault on the controller, the network performance is likely to be affected [18]. With its capability to control network functionality, SDN can enable security services on-demand by providing automated and intelligent services to the cloud service provider and customer requirements. This leads to smooth traffic with flexible traffic management in the controller. The forwarding decision rules are dynamically amended for the network to adapt to the different service updates and application changes in the SDN [19]. However, the majority of contemporary SDN systems only use one SDN controller. As more and larger production networks implement OpenFlow, using a single controller to run the entire network might not be practical for several reasons. Firstly, as switches are added, control traffic flowing to the centralized controller increases. Secondly, when demand increases with network expansion, low setup times might become substantially more problematic since the system is constrained by the managing capability of the single controller. This places a major restriction on the controller's ability to scale up fault tolerance. As a solution to the above, reliable SDN-fault tolerant techniques are necessary and crucial elements for a system to achieve high availability and dependability. However, there are challenges associated with SDN fault tolerance (SDN-FT) that can vary based on the size of the network. In general, small networks struggle with resource constraints, medium-sized networks struggle with scalability and consistency problems, and large networks struggle with complexity and dynamic topology issues [20].

The implementation of SDN-FT has been discussed in the literature using a variety of techniques and approaches. The focus is on enhancing the reliability and resilience of the SDN

architecture to recover from faults and maintain network functionality. This is achieved using different techniques and approaches such as controller redundancy, distributed control plane, network monitoring and detection, fast failover mechanisms, self-healing mechanisms, controller failover synchronisation, fault prediction, and prevention, among others [21] [22] [23] [24] [25]. Overall, the literature on SDN-FT presents a variety of methods to enhance the fault tolerance of software-defined networking. The main goals are reduced network downtime, increased network resilience, and continuous operations even in the event of faults or failures. While SDN-FT has made significant progress, numerous challenges and unsolved problems still exist in the field. Some of these unsolved problems include scalability, controller placement and load balancing, network partitioning, real-time fault detection, predictive fault tolerance, security and resilience, dynamic topologies, and many others [23] [24] [25].

Various approaches and mechanisms in the existing literature address several shortcomings regarding SDN-FT. Therefore, to make the control more robust and to leverage reliability and resilience, this research proposes an SDN fault-tolerance framework system. This system uses various techniques for fault detection, monitoring, and recovery in different types of network sizes. In addition, a proposed architecture using SDN-based techniques for cost-effective on-demand security services is proposed. This can be used by businesses and organisations for their specific requirements and security on-demand requests.

1.3 Research motivation

In recent years, the emerging network paradigm of SDN has gained significant attention. The fundamental principle of SDN is the separation of the control and data planes. This separation enables the development of network applications using high-level abstractions, which are then translated to network devices via a southbound interface. The flexibility and programmability that SDN offers are critical for meeting current network requirements [2]. However, along with its advantages, SDN also introduces new challenges, including a single point of failure, fault management, and controller placement. As the number of SDN-based installations in business networks increases, the need for fault-tolerant controller architecture designs becomes increasingly evident. Several SDN controller-based solutions offer fault tolerance, but these systems are designed with certain trade-offs.

This thesis proposes and designs an SDN fault-tolerance framework for medium to large-scale networks. This framework employs various techniques for fault detection, monitoring, and recovery to address these challenges. Furthermore, this work presents an economically efficient architecture for on-demand security services in SDN. The proposed architecture can be utilized

by a wide range of businesses and organizations to request security services for their IT infrastructure and resources.

1.4 Problem statement

The SDN architecture is composed of three planes: the control plane, the data plane, and the application plane [25]. The controller, which is the primary "intelligence" of the network, is responsible for translating SDN application needs at the application plane into data pathways and providing the applications with a global view of the network's automated program switches [26] [27]. The controller can reconfigure the switches to accommodate changes in network traffic and device failure [28]. Despite the controller centralising the control and management of the entire network, a specific controller can easily become a single point of failure that can impact the entire network's performance [29]. Based on the control plane's orientation, a centralised or single SDN controller is not suitable for a large network due to the risk of a single point of failure, while multi-controllers address the single point of failure challenge but are not suitable for a small network as it introduces a network. Therefore, there is a need to enhance SDN's reliability and scalability. Several approaches exist to address this challenge, but all efforts have been directed towards large-scale SDN.

Contrary to the traditional network in a cloud platform, SDN can offer several benefits to revolutionise the role of data centres by providing flexible ways to control the network functionality in the cloud for vendors, data centre operators, network operators, clients, and end users. As a result, with the SDN updating policy, optimising for a specific application or service is a matter of programming the software, rather than purchasing a new appliance with a registered physical interface. Security can be provided on-demand whereby a tenant can subscribe and unsubscribe from a particular security service at any time according to their requirements. It can also dynamically adjust and replace hardware equipment with software to cut down costs. SDN has the potential to enable cloud-personalised on-demand security to be provided by internet service providers. Clients can choose the specific security services from those service providers that they want. Customers can choose from a list of security services that are offered. Customers can choose whether to use certain security services, and they can set their parameters. An example is that the internet service provider might offer services for parental control or page blocking. Customers can choose to enable it for a specific host or user, for a specific amount of time, or both. The type of websites they want to allow or restrict can also be decided by them. They can routinely see and/or modify those if they are available on their account [29]. Despite the benefits of SDN, new challenges also emerged for the management and control of the network, some of which can be addressed using fault tolerance.

Many existing SDN-FT approaches in the literature still face challenges of availability and high dependency on the system depending on the techniques used. However, some of them may work well in small and medium-sized networks, but their scalability to large-scale networks remains an open challenge [25]. As the network size grows, the complexity of fault detection, recovery, and communication between controllers increases, posing scalability issues [21]. Hence, the issues of optimal controller placement and load balancing across controllers are also a challenge for fault tolerance and overall network performance [27]. Based on the reported state-of-the-art in SDN-FT, there are still several challenges and problems that have not been fully solved in the literature [22] [27]. These challenges and problems include but are not limited to achieving efficient controller failover to ensure minimal disruption and network stability, and optimising resource usage during dynamic topology adaptation by mobile devices, IoT, and virtual networks [27] [30]. While the scalability issue has been addressed in some research to some extent, achieving fault tolerance in ultra-large networks with various types of devices and extensive traffic remains a big challenge [27] [30].

Therefore, in this research, we proposed an SDN fault-tolerance framework system as a novel solution to address the challenge of unexpected faults or failures in the network to ensure its continuous operation. The proposed framework utilises a variety of strategies for fault detection, monitoring, and recovery. In addition, our proposal presents an architecture that leverages SDN-based methodologies to provide cost-effective, on-demand security services. This approach is both flexible and scalable, allowing a wide range of businesses and organisations to request security services tailored to their IT infrastructure and resource needs.

1.5 Research questions

To answer research questions in this study, specific activities were taken, stages were taken, and primary and secondary goals were created to accomplish these goals. To address certain issues that came up throughout the study as well as to pinpoint potential trouble spots, primary and secondary research questions were developed. The research problem addressed in the preceding part served as the inspiration for the study's goal, research questions, and objectives. The research questions that were addressed in this study are presented in Table 1.1.

Table 1.1: Research questions and objectives

Primary research question	Primary research objective
1. How can an effective controller fault-tolerance framework and on-demand security services be designed and developed in an SDN?	Reviewing different literature on SDN, fault-tolerant techniques, and SDN-based security services and proposing an effective FTM in and on-demand security services based on SDN.
Secondary research questions	**Secondary research objectives**
1.1 What is the state-of-the-art of SDN's reliability and scalability challenges? 1.2 How can we model a controller fault-tolerance framework to ensure reliability and improve SDN availability? 1.3 How can the proposed framework address the issues of fault tolerance in both small and large-scale SDNs? 1.4 How can cost-effective on-demand security services be designed based on SDN?	Conducting a comprehensive review of different literature on SDN, its challenges and applications, existing SDN fault tolerance techniques, and SDN-based security services frameworks. Proposed and designed an effective fault tolerance model for SDN that ensures controllers' reliability and availability. Configure the proposed FTM for small- and large-scale networks and implement and evaluate the performances. Proposing on-demand security services utilising SDN functionalities for adoption in the Internet or cloud environment.

1.5 Research aim and objectives

1.5.1 Aim

This research project aims to develop a fault-tolerance model and suggest an on-demand security service, both of which are grounded in the functionalities and environment of SDN.

1.5.2 Objectives

Based on the sub-research questions presented in Table 1.1, this research provided answers based on the following objectives:

- To identify existing SDN fault-tolerant techniques, their challenges and applications, and SDN-based security services frameworks.
- To propose and design an effective fault tolerance model for SDN that ensures controllers' reliability and availability.
- To design and evaluate the performances of the proposed FTM for small- and large-scale networks using various scenarios.

- To propose a framework for on-demand security services utilising SDN functionalities for adoption in an Internet or cloud-based environment.

1.6 Methods of investigation

The RQs and ROs listed in Table 1.1 were addressed using Design Science Research (DSR). DSR is a relatively new research methodology that seeks to solve problems by creating a new reality, rather than attempting to explain or understand the existing one [31]. As a result, DSR aims to create reliability and it can be inferred that DSR has a dual mandate: to use the solutions through insights and theoretical explanations, as valid knowledge is necessary for developing solutions [22]. Horvath proposed that it is possible to carry out a definitive innovative model among experimental and substantiating investigation procedures. The three stages of DSR are as follows: i) the exploration, induction, and deduction of the issues, context, and activities, as well as the formulation of hypotheses; ii) the design and testing of the solution; and iii) the validation of the research, generalisation to other applications, and verification of the hypotheses. To address the research questions and achieve the objectives listed in Table 1.1, this study used a dual mandate, three stages, and a range of interactions.

Moreover, researchers in [27] [30] [32] have emphasised these interactions and phases in the following sequence: i) defining the issues and establishing goals for a solution; ii) planning and creating artefacts; iii) demonstrating the issue's resolution by utilising the artefact; iv) assessing the solution by contrasting the stated goals and the actual results obtained from using the artefact; and v) communicating the issues, the artefact, and its usefulness and effectiveness to others. We make use of this six-step developmental chain and divide them into three interactions to design the proposed framework, with each interaction being broken down into a distinct set of phases. This idea was supported by [33], who claimed that researchers typically move outwards from the point of entry of the research and do not always have to start from the first step. Instead, they typically travel through all the processes in one manner or another. Consequently, details about the research design and methodology are discussed in Chapter 3.

1.7 Research contribution to knowledge

This section presents the possible contributions to the knowledge of the research performed in this thesis.

i. The thesis proposes an SDN controller fault-tolerant framework model that uses a modified controller that uses a built-in stationary agent (SA) that monitors the status of controllers in the network.

ii. A proposed SDN controller fault-tolerant framework that has a stable database model with a modified controller which uses a built-in stationary agent (SA) to monitor the status of controllers in the network.

iii. The proposed SDN·FTM system optimized network performance with Open· Flow controllers having a low latency rate compared to Ryu's controllers and this demonstrates that the controller position selected was best for the performance of the network.

iv. Provides recommendations for future works to improve the network dependability, resiliency, and security application.

1.8 Research output

In addition to the knowledge-based, below are possible outputs from this research study:

1) Two conference papers and presentations whereby some section of this research work was prepared and presented at two international conferences; as well as published in the conference proceedings [12] [34].
2) An original scholarly research article was prepared from this work and was submitted for publication in an accredited academic journal to disseminate the results of this study in the academic body.

1.9 Thesis organisation

This thesis is organised as follows: Chapter 1 presents the introduction, problem statement, and motivation for this study, along with the presentation and explanation of the research aims and questions. Chapter 2 covers various types of literature used in conducting this research, which includes conference articles, journal articles, reports, magazines, as well as other student work within the context of this study. Furthermore, various comparisons of different fault tolerance frameworks and simulation tools were conducted to evaluate their design simulation platform and performance against the proposed SDN-FTM.

Moreover, Chapter 3 explains the research design and methodology used to develop and evaluate the proposed SDN fault-tolerant framework. Additionally, it describes various steps and interactions in the development and evaluation of the proposed SDN-FTM as well as the

proposed on-demand security services. Finally, a sequence of design principles, components, and mechanisms for the development of the artefacts are also presented, showing how they are related to the approach used in the development of the system. Chapter 4 presents various assumptions taken to ensure that the design system is fault-tolerant, as well as diverse design principles and requirements that led to the choice of the FTM. Furthermore, it also presents the architecture of the proposed FTM, as well as the different components and their functionality. The implementation and evaluation of the proposed SDN fault tolerance Model using the architecture design in Mininet are presented in Chapter 5, using different scenarios and customised parameters that suit the design principles. Furthermore, Chapter 6 presents controller placement and SDN FT evaluation using various experimental setups and system properties for the simulation. Chapter 7 introduces the proposed on-demand security services, which are based on SDN and Network Function Virtualization (NFV) technologies. This chapter also outlines the requirements necessary to implement the capabilities of the proposed architecture. Finally, Chapter 8 provides a comprehensive summary of the entire thesis, reflecting on the objectives of this research.

1.10 Summary

This chapter provides an overview of the current research, beginning with the background of the research. It discusses the challenges brought about by the use of various devices on the internet and how SDN emerged as a new paradigm to overcome these challenges. Further, it also indicates how SDN-fault tolerant techniques are necessary and crucial elements for a system to achieve high availability and dependability in the event of network challenges such as faults and errors. To address this problem, a framework system was proposed as a novel solution to tackle these challenges of unexpected faults or failures in the network, ensuring its continuous operation. The benefits of fault tolerance were presented as a motivation to offer several SDN controller-based solutions. Furthermore, the research questions being answered in this study were presented as a means to achieve the research aims and objectives. In addition, DSR was identified as the most suitable method to use in this study to achieve the research outputs. Finally, the chapter concludes by outlining the organisation of this thesis. A literature review is presented in the next chapter.

Chapter 2

Literature Review

2.1. Introduction

With the exponential growth in internet usage and modern technologies, traditional networks are often seen as static and highly inflexible, particularly when handling dynamic network traffic [17] [35]. This fundamental challenge and network complexity have been recently addressed by the advent of a cutting-edge networking model known as SDN. In the current era of interactive and dynamic multimedia streaming, SDN has proven its ability to centralize network control, management, monitoring, and configuration through network programmability techniques [12]. SDN is designed with the capability to automate network operations and features an architecture that separates the control logic from the forwarding device. The dynamic nature of today's applications, which are evolving on cloud computing technologies, serves as compelling evidence that SDN is ideally suited to address the multitude of issues that have plagued traditional networks over the years [17]. As a result, SDN is engineered to facilitate the development of the next generation of networks, offering several adaptable, cost-effective, and innovative opportunities [12].

The control plane, the primary component of SDN, houses the SDN controller. This controller is responsible for managing and directing the entire network, serving as the central hub for decision-making [12]. It acts as a bridge and a conduit for communication between the application and data planes, and other planes. By offering an abstract perspective, providing network devices with appropriate forwarding decisions, distributing infrastructure, sharing applications, and granting network access to end-user devices, the controller organizes and governs the network's behaviour. Additionally, the architecture of the control plane has considered various architectural orientations of the controller, such as single-centralized or multiple/distributed [24] [32]. To ensure the scalability, dependability, and availability of the network, both single centralized and distributed systems are oriented physically and intellectually [12]. However, managing a large-scale network with a single controller is highly challenging and leads to a single point of failure [12] [24] [32]. It is necessary to determine the number of controllers, their location and distribution, their cost, and any additional issues resulting from the deployment of multiple or distributed controllers [33]. The CPP, which addresses this location-allocation issue during controller deployment, has grown in importance

in SDN research. It has been determined that the CPP is the optimal technique to leverage the multi-controller deployment in the SDN environment [36] [37] [38]. Several strategies have been developed and proposed in the literature for the CPP, each with its objectives, advantages, and disadvantages.

2.1.1. Outline

As a result, this chapter evaluates various literature, different existing fault tolerance frameworks, and different simulation tools, and was conducted to evaluate their design simulation platform and performance against the proposed SDN-FTM and some of the existing CPP by classifying, analysing, and contrasting their methodologies. Understanding their tactics and offering potential lines of inquiry for subsequent research to improve tactics are the goals. The study provides valuable information and insight into the significance of CPP approaches concerning dependability, in a multi-controller scenario, load balancing, resilience, and network state consistency [12].

2.2 SDN Overview

A network paradigm known as SDN allows for centralised network monitoring, by separating the control plane from the data plane [39]. Figure 2.1 depicts the three planes that make up the SDN architecture: application, control, and forwarding [39]. The SDN controller, acting as a network operator in a traditionally centralized organization, is responsible for transferring requirements from the SDN application layer to the SDN data routes. It also provides the SDN applications with an abstract view of the automated program switches [40] [20]. In other words, the controller can establish connections between nodes, provide flow rules, forward tables, and ensure their maintenance [41]. Furthermore, every network component is visible to the controller. On the other hand, the data plane accommodates the network hardware or forwarding elements that connect to the controller via the southbound interface [12]. These elements operate according to the rules specified in the controller's orders, directing them to their next destination. The forwarding plane includes both physical and virtual nodes that have contacted the network [42]. In addition, the applications and services that determine network behaviour are the responsibility of the application plane. Applications communicate directly, programmatically, and openly with the controller via the northbound interface to convey network needs and requests [42]. The basic idea behind SDN design is to provide programmability on the control plane while separating a controlled entity from a controller entity. Three planes make up the SDN architecture: the application, control, and forwarding planes. According to Figure 2.1, these multiple planes are connected by a variety of control plane interfaces.

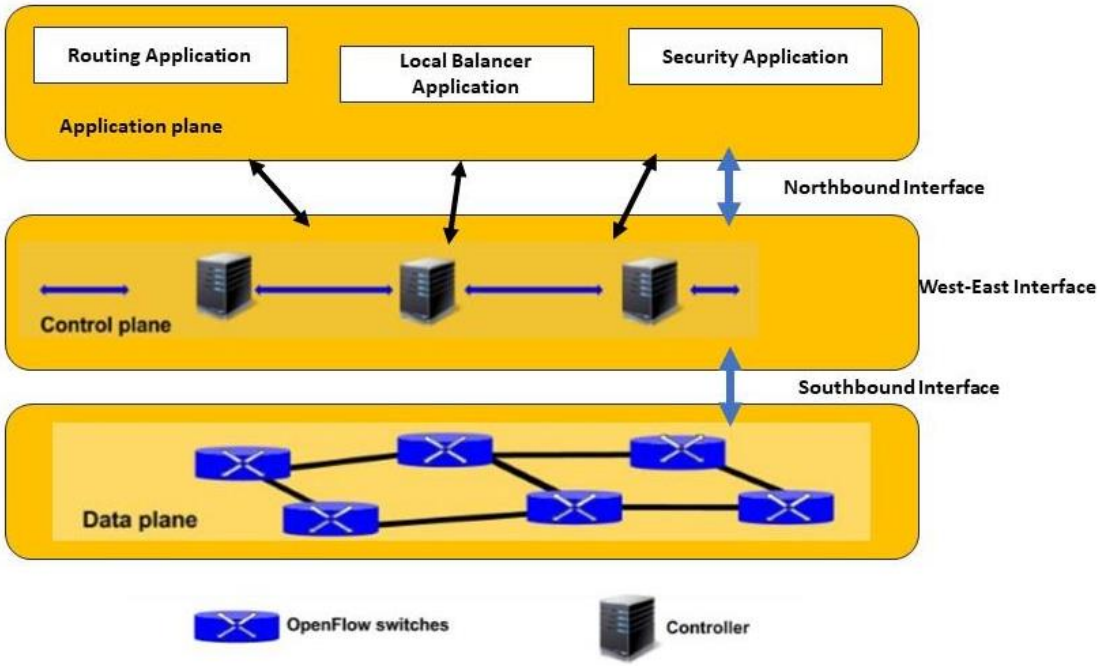


Figure 2.1. SDN architecture [28].

- Application plane: This is the location of the programs and services that specify network behaviour. The control plane's routing procedures, for example, are not regarded as applications because they directly (or mostly) support how the forwarding aircraft operates. The application plane also handles special tasks that require much closer inspection [28].
- Control plane: Makes intelligence decisions on how data transportation or how one or more network devices should forward packets. The main duty of the control plane is to establish and define the neighbourhood topology of the network by focusing more on the device's forwarding plane and less on its operating plane [28].
- Forwarding plane: This is in charge of directing packets in the data stream to their subsequent location following instructions from the control plane. Among the many functions that the forwarding plan can execute are forwarding, discarding, and modifying packets. For control-plane services and applications, the forwarding plane serves as the destination. The forwarding plane, often known as the 'data plane' or the 'data path' might include forwarding resources such as classifiers [28].
- Northbound application programming interfaces (APIs) are the southbound APIs in SDN to establish communication with the applications and business logic application

plane, as shown in Figure 2.1. Northbound APIs also assist the network administrator in programmatically configuring network traffic and the deployment of services [28].

- Southbound APIs: The use of southbound APIs in SDN software-defined networking is among the most common protocols used in OpenFlow to pass on information to the switches and routers below in the forwarding planes [28].

Northbound application plane interfaces are used to connect the application plane and control plane that contain security systems, monitoring services and some other general applications that require access to network devices. The control and data plane, on the other hand, are linked through southbound APIs, as shown in Figure 2.1. To increase flexibility for network control and administration, SDN design separates the implementation of the network control logic forwarding activities [27]. When compared to traditional networks [29], where the control and data planes are more securely coupled and traditionally function in a completely distributed manner, the main goal of the SDN architecture is to make the deployment of new control plane functions, such as routing strategies, simpler.

A component of the SDN control plane called an SDN controller oversees flow control to enable intelligent networking. Application for SDN communicates their desired network behaviour and network requirements freely, directly, and programmatically to the SDN controller via Northbound interfaces (NBIs) [29]. Fundamentally, protocols like OpenFlow, which enable servers to direct switches where to deliver packets, are the foundation of SDN controllers. The controller, the central component of SDN design, carries out the functions of a network operator in a conventional network. The controller, a logically centralised entity in a reliable SDN environment, facilitates the seamless translation of SDN applications' layer requirements into actionable data pathways, while concurrently endowing SDN applications with a conceptualized perspective of the network's automated program switches [31]. The target application’s specific goal can be achieved by the SDN controllers thanks to dynamically designed programming logic for packet forwarding, network switching decisions, and network routing [5]. The controller can change the switches’ configuration to accommodate changes in network traffic and hardware faults [31]. Although its main role is to centralise control and monitor the network, a particular controller can easily become a solitary point of failure that can lead to entire service interruptions or incorrect network behaviour [24].

A programmable data plane, which manages packet forwarding based on instructions from the control plane, and a centralised control plane, which offers a global perspective of the network, are introduced by SDN as a novel way of managing the network. In conclusion, SDN gives a

fresh perspective on network management that increases adaptability, efficiency, and flexibility. SDN opens the door for future networking advances by enabling centralised management, network virtualisation, and programmable behaviour by separating the control plane from the data plane. To fully enjoy the advantages of SDN, security, scalability, and interoperability must be carefully considered.

2.3 Security services in the cloud

Over the past decades, the networking landscape has changed as never before. With the integration of cloud computing technologies, pay-as-you-go consumers or users can use and implement IT services while avoiding significant capital expenditures on their IT infrastructure. Consequently, in terms of network infrastructures, cloud computing presents the IT industry with significant prospects, data storage, operating systems, applications allocation, resource scheduling, transaction management, bandwidth allocation, memory management and many others. However, the openness and virtualisation featured in the cloud environments make the issue of security and protection a challenge [16]. Cloud networking architecture allows users to specify bandwidth requirements for application resources hosted in the cloud to ensure good performance for on-demand service deployment. Overlapping applications in the cloud requires some guaranteed bandwidth between physical devices such as servers and switches to gratify effective user movement on the network. Therefore, the virtualisation features in the cloud environment are becoming a challenging task for network operators in data centres, cloud service providers and users in the IT industry to maintain security. These challenges are posing and raising numerous concerns in maintaining network traffic, access control and software verification for service on-demand [16]. In terms of network infrastructures, cloud computing thus presents the IT industry with significant proposals. However, many privacies and security long-standing one in distributed computing, many privacies protection, and preservation technologies exist and are being used directly in the cloud context [16].

Secure computation was developed by researchers in [29], as a privacy protection and preservation technique that utilised encryption to link a group of unreliable nodes to cooperate on a job. The technique works well for a client's hardware interface to secure compute channels while protecting private data or information in cloud services. To address security issues at the cloud application platform, the programming platform MapReduce has been explored to address some privacy threats [39]. Virtualisation, which primarily focuses on how to virtualise or isolate sensitive information based on secure hardware, is the key aspect of privacy protection and preservation in the cloud environment [40]. At this level, privacy protection and

preservation must essentially take better virtual machines based on current cloud services into account. The issue at hand is that most conditions in cloud computing services level agreements (SLAs) primarily disaster recovery and business continuity are service-related, such as performance and availability of services [40].

The confidentiality, integrity, and availability of data and applications housed in cloud environments depend on security services in the cloud. Numerous advantages of cloud computing include cost savings, scalability, and accessibility, but it also presents particular security challenges. These issues are addressed by integrated security services, which also offer a strong defence against growing cyber threats [39]. To sum up, incorporating strong security services into cloud environments is essential for defending data, apps, and users from an ever-growing variety of cyber threats. These services give businesses the knowledge, expertise, and adaptability they want to maintain a strong security posture and guarantee a secure and dependable cloud computing experience. It is essential to invest in comprehensive security services to safeguard sensitive data and uphold client confidence as cloud use keeps increasing.

2.4. SDN-fault tolerance

Traditionally the open nature of the architecture for cloud-based on-demand services raises concerns over access control and service verification on the network. For example, in an ad-hoc network in a cloud-based environment, there is no solution to control the exchanges between various devices on the network [15]. However, in SDN architecture, different nodes on the network communicate with the SND controller to provide them with the appropriate protocols for forwarding decisions. Nevertheless, since the controller is the core of SDN architecture, if there is a fault in the controller, it affects the entire network's performance. Therefore, sustaining a regular controller state in an SDN environment becomes a challenge. There is a need to include some mechanisms to ensure that controllers send forwarding decisions consistently to handle the switch state of the controllers.

Many different fault-tolerance techniques exist that are used to address this form of consistency in both controllers and switches in the SDN platform to offer complete transparency to the network applications and to make the controllers more vigorous [2] [5] [18]. Currently, many fault strategies are offered by the SDN controllers that can be found in an SDN platform. However, there are some unresolved problems, particularly concerning the trade-off between consistency and performance in a fault-tolerant SDN architecture [32]. To address part of the problem [32], some industrial SDN controller resolutions integrate fault tolerance into the design of controller architecture [33] in their system. However, SDN fault tolerance deployment

covers various fault domains, such as switch failures in the forwarding plane, and failure of the switch-controller connection in the control plane and the controller itself. One of the existing research projects focused on addressing issues in the forwarding plane [36] and created a fault-tolerant SDN architecture that could graduate to a bigger network and quickly recover from many faults [33]. Regarding the viability of reliable and consistent data stores for SDNs, [36] designed a strong fault-tolerant SDN controller framework that provides highly available network resources and a strongly consistent data store to achieve better performance.

Despite its growing popularity, SDN-FT enables network flexibility by clearly separating the control and data planes, making network management simpler [43]. At each level of its architecture, SDN is vulnerable to new failures and issues, as many academics have noted in the literature. The centralised controller must be fault-tolerant to maximise performance and maintain the dependability and high availability of small to medium networks. To eliminate the single point of failure, errors should be found and repaired as rapidly as feasible. This is because the controller is the network's primary building block and having just one controller significantly affected the operation and availability of the network. The capacity of the system to continue working in the event of a defect, which can include hardware, link, or networking failure, is described in the literature as a fault. In SDN the fault-tolerant ensures network functionality despite an event of a failure or errors. The issue of failure in a conventional network is linked to a link or node only. In SDN, software flaws or hardware issues are the main causes of failures and defects [43]. Due to the network's unreliability, which is characterised by frequent failure, recovery of links, failure of a node, and software fault. The main factor leading to a network failure is a fault, whereas a failure happens when the network is unable to perform its functions correctly. Mechanisms such as fault localisation, fault recovery, and fault detection make up an error or fault-controlling procedure that assists in the way to address issues of being fault-tolerant [30].

To address failures, fault management mechanisms typically depend on distributed protocols such as OSPF, which causes a variety of problems. A single failure may result in many alerts since a failure may be detected by various systems. Transient failures may cause network device inconsistency, and after a failure, the state of the entire network may take some time to stabilise [30] [44]. In such cases, the information on the network condition may be inaccurate or incomplete because of network partitioning during the process of failure. A global network perspective expands the opinions for fault localisation; for instance, SDN is being used as a unique technique to investigate solutions to numerous fault management issues that exist in

today’s dynamic networks. SDN’s logically centralised and programmable interface is utilised [45] [46].

Fault tolerance in SDN environments is essential to maintaining the dependability and accessibility of network services [47]. SDN fault tolerance improves network resilience and reduces downtime by efficiently managing faults and interruptions. SDN fault tolerance is essential for maintaining the dependability and accessibility of network services [35] [47]. Networks may handle defects gracefully using SDN's centralised control, dynamic reconfiguration, load balancing, and redundancy, minimising interruptions, and downtime. Organisations can create robust and resilient SDN designs that can survive unforeseen difficulties and maintain a high degree of network performance by utilising automation, testing, and monitoring.

2.4.1 SDN fault tolerance mechanism in distributed systems

In the event of a fault or error, to prevent network service failure, several strategies are being deployed [35]. To implement fault tolerance, systems recovery mechanisms and error detection are used. Unlike recovery, where the state of the system is changed due to one or more faults to a state without recognisable faults that may be reactivated, the evidence of mistakes is determined through error detection. The following are the two recovery methods: while fault handling recovery prevents faults from re-activating, error handling recovery removes errors from the system state [22]. Moreover, the underlying fault assumption informs the technique selection for detection and recovery [22]. Therefore, SDN fault-tolerance mechanisms are studied within the context of study in the area that improves fault-tolerance within the context of SDN networks.

Various mechanisms are used to address fault tolerance issues in distributed systems, among others, the most useful are replication-based fault tolerance techniques, process-level redundancy techniques, and fusion. For this research, the focus is on the replication-based fault tolerance technique. One of the most often used techniques is replication-based fault tolerance because this method replicates the data on various additional systems [23] [24] [25]. A request can be issued to one replica system while the other replica system is still receiving it using the replication mechanisms. In this manner, if one or more nodes malfunction individually or collectively, the system does not cease to operate. A system gains redundancy through replication. The replication protocol has several stages, including client contact, server coordination, execution, agreement as well and client response. Some of the most important concerns in replication-based mechanisms are the issue of consistency, degree of replica and

on-demand. To ensure consistency in the replication process, multiple copies of the same entity updates can be made available by any user on the network. On the other hand, the degree of the replica can make use of procedures to replicate data using a certain protocol. Active and passive replications are the two major types of replication techniques. The primary and backup systems are in active mode controllers execute switch requests simultaneously, which could lead to inconsistencies between the two controllers [48]. The network has one major controller, on backup control, and two disjoint paths [48]. To address the issue of active replication, [49] execute an active replication approach in the Ryu controller and provide consistency among the distributed controllers using the OpenReplica service. The Revana-tolerance SDN controller platform, which handled the control messages, was introduced by [28]. Nevertheless, in order, to enhance the active replication process, all systems relied on an external service or new protocol. As a result, it is difficult to implement their strategies, and doing so risks lowering the controller's computing performance [28] [49]. A switch only connects to one controller in a passive replication scenario, and the controller reacts to requests and updates additional controllers. To increase SDN resilience, [28] presented a novel approach that made use of the component organisation of CPRecovery. The processing expenses of slave controllers are reduced compared to the active mode. To ensure consistency in the case of breakdown, the whole slave controllers must, however, give the master controller monitoring [28] [49].

Distributed systems' SDN fault tolerance methods are essential for preserving network stability, guaranteeing continuous communication, and reducing the effects of failures in intricately coupled environments. Fault tolerance is even more crucial in distributed systems due to the several nodes and components that are dispersed across various places [31]. It is possible to handle defects in SDN settings in a comprehensive manner because of the division of fault tolerance in SDN into proactive, reactive, and hybrid techniques. Thoughtful consideration must be given to concerns including overhead controller resilience, coordination, scalability, security, and interoperability when establishing fault tolerance in SDN systems [24] [25]. Network administrators may create strong, dependable SDN infrastructures that can resist errors and deliver consistent, high-quality services by addressing these problems.

2.4.2 Classification of SDN fault tolerance and possible issues

In the literature, numerous articles have examined problems with or attempts at fault management in SDN. These researchers vary in their discussion breadth and how they categorise fault management initiatives [25] [48]. A framework for fault tolerance has been

presented in [23], which includes a fault manager with many fault-tolerant components that monitor and detect network faults using heartbeat signals and recover from failure using checkpoints. In this approach, only one controller handles the network control in a passive replication system, and in the event of a failure, a voting technique is used to allocate a new controller to take charge of the network. Researchers in [24] and [25], offered insights about fault tolerance and reliability challenges highlighted by SDN while researchers [50] and [48] offer a classification of the research around SDN in the literature. The former groups assessed works into resilience disciplines according to [49], whereas the other groups' efforts were divided into categories based on the specification fault management attributes such as fault detection and fault recovery. In [50] [49], a discussion on SDN fault management provides an overview of SDN fault management while concentrating more on problems related to each SDN architecture layer. They also reviewed overarching strategies, compromises, and significant contributions, and research gaps were also identified. They also covered optical and wireless networks in their discussion of SDN fault management challenges. In their study of existing SDN fault management options, [51] included a discussion of SDN fault tolerance as well as SDN fault management with a focus on system monitoring, fault diagnosis, fault recovery, and repair. To conclude their study, they compared and examined the available SDN fault management technologies around research done between 2008-2017.

The traditional fault tolerance strategies were discussed, and their linkages to SND were examined in [52]. A comparison of link/node data plane failure detection and recovery technologies with a brief discussion. They also contrasted recovery and protection strategies for link/node failure and talked about conventional fault-tolerance methodologies [52]. provided a survey on resilience research in SDN, focusing on six disciplines [53], incorporating dependability, security, performance, and traffic and disruption tolerance. They highlight which resilience traits are covered by each discipline and give a summary of the approaches in each category. Additionally, they defined procedures in terms of the SDN plane to which each attempt belonged.[35] concluded by summarising discovered issues and difficulties considering several domains and resilience disciplines. There is no comparison of the methodologies surveyed. The interrelation between fault tolerance and SDN is examined in [53]. They describe how fault tolerance works on conventional networks and what aspects of the OpenFlow protocol could be used to give the network fault tolerance. Then, categories of methods are created based on the relevant plans and the specified fault tolerance attribute. The failure recovery and links/nodes failure detection methods in the data plane are constructed. The authors in [30] cover techniques for handling controller and control channel failure on the

control plane. There is no comparison of different control plane methods presented in [30]. There is no explanation of the data plane's other components, such as the application plane.

Most fault tolerance problems occur at the levels of each layer, and some of them cause layer interaction. As a result, while classifying layers, layers that might or might not be affected by a certain failure type or that might be employed to address the issue are not considered. For instance, the failure of a link connection network devices to the controllers rather than being a part of the control network may or may not influence the control and application layers. For this study, we discussed issues in the infrastructure and control or infrastructure interface. Controller and link failures occur at the infrastructure layer and the interface between infrastructure and control, respectively, are consequently classified differently in the network concept. We adopt a bottom-up strategy, moving from the infrastructure plane to the application plane. Traditional networks are more susceptible to problems such as link and node failure, which are mostly to blame for infrastructure layer fault tolerance issues. However, centralised management and programmability ability using SDN cast a new light on those problems, enabling creative solutions. Additionally, in conventional networks, fault location and detection must be carried out decentralised. By maintaining a centralised network view that all network devices must update in the case of link failure, SDN programmability can make this simpler. A controller could, for example, determine which switches are impacted by a particular failure and notify only those switches [52]. However, since there was more communication overhead, recovery time was slower, and therefore it must be properly planned. However, a botched restoration could cause several issues, such as overtaxing network controllers and prolonging the recovery period.

Hybrid SDN techniques allow for the coexistence and communication of numerous protocols and processes to improve failure recovery. Since the network view may include traffic engineering applications, quality of service (QoS) features and congestion-aware routing, centralised control provides unique alternatives for network recovery from link and node failures. Analysing the features of different network configurations is required to offer fault tolerance to a variety of scenarios, such as optical networks, smart grids, and data centre networks. The infrastructure and control interface of network devices and controllers are intrinsically dependent on one another for proper functioning because network devices and control logic are physically separated. The need for a larger network is also increased by centralised management since links that transmit data and control traffic have different operational priorities.

Table 2.1: Summary of SDN fault tolerant issues

Ref.	Challenges	Focus layer
[24]	Failure isolation, fault tolerance state and placement	Data Plane
[25]	Monitoring, detecting, and recovering from failure	Control Plane
[28]	Fault tolerance and reliability	Control Plane
[52]	Fault tolerance and reliability	Control Plane
[25]	Several domains and resilience disciplines	Control Plane
[30]	Fault detection, fault recovery and optimisation in wireless networks	Architecture layer
[49]	Fault tolerance and resilience	Control Plane
[50]	Fault tolerance and resilience	Control Plane
[30]	Fault detection and fault recovery	Control Plane
[52]	Monitoring, identifying, fixing, and recovering from faults in the system	Control Plane
[49]	Systems for link/node failure detection and recovery, as well as strategies for link/node failure restoration and protection	Control/Infrastructure Layer
[54]	Resilience	Control Plane
[30]	Domains and resilience	Control Plane
[49]	Failure recovery and links/nodes failure detection	Control Plane

In SDN, controller-switch communication failures and control network failures are similar to link or node data delivery failures, respectively. The placement of controllers and control failover are major related SDN problems that are being discussed [24]. Table 2.1 presents a summary of the SDN-fault-tolerant issues.

2.5. SDN controller placement

To address the issue of where and how these controllers are put in an SDN environment, CPP is a problem of multiple-controller deployment in a big network [24]. To decrease service overhead delays and boost network performance, the best number of controllers needed to be placed in the ideal location. The performance of the SDN network is significantly impacted by CPP, which has recently attracted a great deal of scholarly investment. By redefining CPP as an optimisation problem, it can be resolved. To do this, the SDN network is modelled as an undirected graph, G, represented by G = (V, C), where B = V denotes a controller, and C = { C1, C2, C3, …CN} and C ⋹ V [24]. The idea placement, or the distance, d (d_i) between network nodes that create the shortest paths between the controllers and switches that must be located, is therefore found to address this problem [24] [54]. To obtain good network performances and QoS, propagation, and processing latencies must be reduced, and resilience, energy, etc. must be maximised.

Additionally, the existing CPP methodologies have been widely divided into capacitated (CCCP) and incapacitated (UCPP) approaches [54] As indicated in Figure 2.2, these two categories are further divided into subclasses. In terms of controller performance in the SDN environment, the UCPP category signifies a limitless capacity. To put it another way, the controller frequently ignores the pressure placed on them and does not take capacity into account to be a performance restriction. In this case, it is essential to deal with the CPP while taking the controllers' load into account. The majority of contemporary CPP solutions assumed that all switches on the network had fixed and specified loads, ignoring the load or capacity of the controllers as a limitation. To reduce the average latency from switch to controller, the authors in [54] defined UCPP as a minimum k-medium problem and a minimum k-centre problem, respectively. The CCPP category includes several controllers that, when deployed on a network, consider the load or capacity of the controllers to prevent controller failure. The argument is that while certain controllers could be routinely overworked, others might not be. As a result, the distribution of loading must be balanced. There are two ways to determine the load on the controller: the amount of flow in the network per second and the number of switches k = linked to that controller. During placement, the controllers' capacity is considered by using network partitioning-based CCPP, these types of CCPP exist as static traffic-aware, dynamic traffic-aware, and fault-aware. Load balancing was employed [54] to disperse load to the controller with a minimal impact on the network.

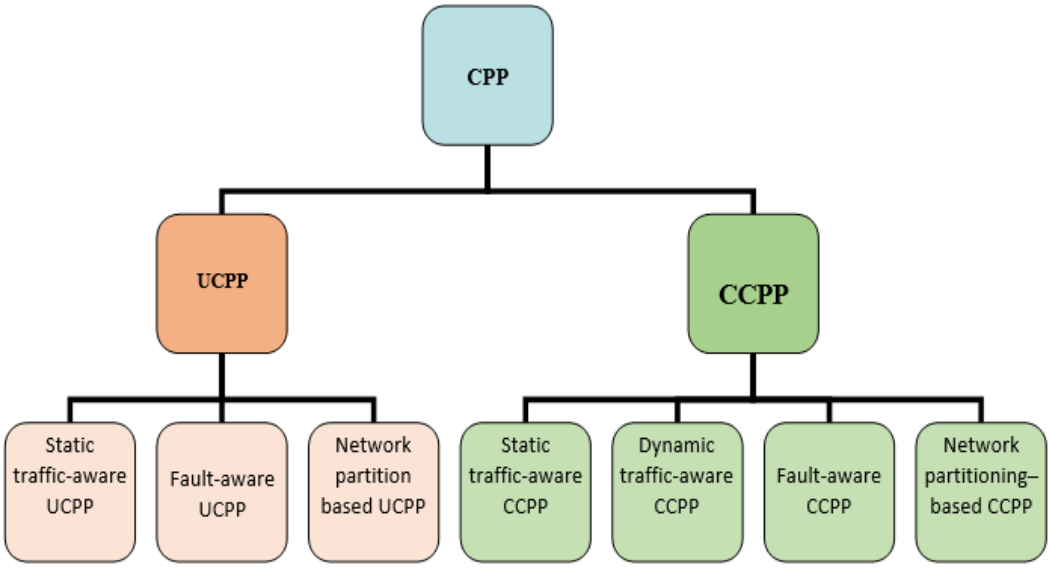


Figure 2.2: Summary of CPP classification [12]

2.6. Analysis of controllers' placement schemes

Numerous studies have shown that in large-scale SDN, the primary difficult aspect affecting network performance is SDN and CPP. In this section, we analysed several CPP approaches currently in use and provided analyses of both the capacitated and incapacitated CPP categories [12].

2.6.1. Capacitated CPP

This section explores the CPP systems of current works that incorporate load balancing and other significant metrics in their objective function [12]. The studies discussed in [12] conducted a preliminary survey before proposing a CPP system. This system employs a clustering-based technique grounded on affinity propagation to enhance the network capability of the controllers, thereby reducing propagation latency between switches and associated controllers. The authors devised a clustering-based network partition algorithm, known as the Shortest Path Distance (CNPA), to identify patterns in the network capacities of the controllers and enhance performance. This technique partitions the network, thereby reducing end-to-end latency. The method was evaluated using the Internet2 OS3E topology and compared against the k-means and k-centre algorithms. The results indicated that CNPA outperformed both the k-centre and k-means in terms of maximum latency [55]. CNPA, an enhancement over CPNA, was further expanded to create clusters and determine the number of controllers. This was generally done to reduce propagation latency. By employing CNPA, the maximum end-to-end latency between controllers is minimized, and controller queuing latency ensures that a substantial number of controllers are placed in the subnetworks within an acceptable number of groups. Simulations were conducted to assess the effectiveness of the method using the topologies of the Internet2 OS3E and ChinaNet. The results suggest that, compared to the k-centre and k-mean methods, the maximum latency between controllers and switches can be significantly reduced.

The improved exemplar-based clustering method, which leverages affinity propagation, has been developed by [56] as a CPP strategy for SDN. This method aims to reduce the propagation delay between controllers and associated switches. Instead of setting them a priori, the CPP system selects the ideal number and position for controller placement based on the network's structure. The effectiveness of this method in terms of stability and precision in controller placement, which reduces latency, was demonstrated by simulating the scheme's performance using the OS3E topology and comparing it to the k-centre and k-means algorithms.

Researchers in [57] proposed a method for installing controllers in a Virtualised SDN (VSDN), known as VCPP. This method simultaneously optimizes both hypervisors and controllers in the VSDN, also known as Joint Hypervisor and Controller Placement (JHCP), to decrease the worst-case latency between network nodes. The methodology was designed to reduce average latency [57]. Evaluations were conducted using two Internet Topology Zoo simulations and compared to current models of the Hypervisor Placement Problem (HPP). The results demonstrated that both VCPP and JHCP outperform k-HPP in terms of worst-case latency, average latency, maximum average latency, and average maximum latency [57].

In a study by [58], a network partitioning technique known as the CPP scheme was introduced. The authors in [16] proposed a Density-based Controller Placement (DBCP) method, which employs a density-based switch clustering technique to divide the network into multiple sub-networks. The goal is to maximize the number of controllers subject to a capacity constraint by allocating a single controller to each subnet. In DBCP, the subnet size is determined by the controller's capacity, and density-based clustering is used to identify the optimal number of controllers. According to the evaluation of 265 topologies, this technique outperforms other CPP cluster-based systems like k-centre and POCO in terms of fault tolerance, time consumption, and propagation delay. Furthermore, a controller's capability in CPP is identified as a key element linked to the resilience of the control plane [58].

Contrary to most previous research, which views the problem of network traffic flow as best managed by switch controllers or intern-controllers, [56] presented a CPP technique to address placement and controller allocation. The authors selected the optimal controller placement and switch-controller assignment by considering propagation latency, hop count, and link usage. Moreover, the strategy introduced a novel Analytic Hierarchy Process (AHP) method, called the Controller Placement Generic Algorithm, to address the issue of QoS in the SDN network environment. This method considers the impact of various variables on switch assignments in CPP. In experiments using the Internet2 topology, the proposed method outperformed previous CPP methods in terms of placement, latency, and load balancing. This strategy is compared to one described by [58], which is based on the current number of controller failures. The goal is to maintain the mapping of switches to a predetermined number of controllers, where each switch provides a percentage of demand to each of its assigned controllers while minimizing network costs. In addition to adjusting the total number of controllers in the SDN, the authors suggested a methodology that reduces worst-case and average controller latency as well as enhances dependability [58].

2.6.2. Uncapacitated CPP

The infinite measure of controller performance was examined in the SDN CP studied in this section to cope with load and capacity during controller installation and prevent controller failure. To increase dependability, [56] developed a CPP method based on network portioning. The optimal number of controllers and their placement path loss were determined by the authors using the propagation latency approach, and a set of routes defining how the switch controllers communicate with one another was created. The plan minimises control path loss while maximising controller reliability by automating placement selections with several established algorithms. Through simulations with actual topologies, the approach's efficacy was assessed. According to the results, the SDN controllers' reliability has significantly increased, and placement performance is solely dependent on the particular methodology used. Additionally, the suggested CPP technique highlights how the availability of controllers affects the network's dependability, even though load balancing was not considered, and the simulated annealing solution was almost perfect. In essence, having too few or too many controllers can compromise reliability. The authors considered both reliability and average latency as performance metrics. The effectiveness evaluation of the strategy reveals that it is not possible to optimize reliability and average latency simultaneously. The strategy encompasses propagation delay, installed controllers, and controller placement based on the network's shortest path between switches [59]. It also considers the common delay between assigned controllers and switches.

Furthermore, a novel strategy for addressing the CPP was introduced [7]. The authors scrutinized the key issues related to SDN-CPP by partitioning the SDN network environment into multiple domains and assigning each controller to an appropriate domain [7]. This method was devised to balance the load on the network controllers. By constructing a common usage measure for optimizing the controller based on the shortest path between nodes in each domain in the SDN network environment, a greedy technique was developed for this study to connect each domain to a controller [7]. This study contrasts with [59], which focused on enhancing the CPP rather than a specific measure by identifying the optimal position for the controller.

According to this paper [60], an innovative architecture was proposed for placing multiple controllers, and the fresh problems with dynamic controller provisioning in WAN were both highlighted. The method focuses on sustaining the SDN network platform while establishing a dynamic connection between several actively adapted controllers. Similar to [61], the greedy knapsack method was also applied to lessen the overhead of the current network configuration between controllers and switches. To tackle it, the authors created a framework and heuristic

algorithms, DCP-GK and DCP-SA, to maximise network accuracy and performance. When simulating, two distinct topologies were used. The outcomes demonstrated that the architecture successfully balanced message overhead and flow time. Additionally, DCP-SA performed better than DCP-GK although with slower convergence. Research by [61] suggested controller backup capacity, and for full resistance against a predetermined number of controllers, switches to controller latency failures to reduce network costs. The authors switched to mullet-controller mapping methodologies and used three placements. When compared to other existing techniques, with less backup capacity, the suggested architecture achieves controller resilience [59]. Similar to this, authors [59] used graph theory's clique-based approaches to solve the challenge of balancing the traffic load of switches utilising a polynomial-time complexity and heuristic evaluation. The effectiveness of various solutions was examined, and the results demonstrated these solutions' efficacy in a variety of wide-area networks, and the results demonstrated that they could reduce expected control path loss.

One of the objectives of this research is to determine the ideal placement for controllers' deployment in terms of their position and location to improve controller performance during fault tolerance. For the evaluation of the proposed FTM, the deployment of the controller in the small-scale as well as medium, to large-scale network UCPP was selected as the best approach, because it considered load balancing and other crucial metrics used by the controller. Therefore, a deep comprehension of this CPP was essential to this investigation. Table 2.2 lists the algorithms used, the methods used, the measurement or criteria taken into consideration and the approaches' goal.

Table 2.2: Summary of CPP Schemes [12].

Reference	Algorithms	Procedures	Classification	Type of network	Factor Considered	Objectives
[59]	Algorithm for partitioning a network based on clusters	Network portioning Simulations, Internet2 OS3E topology	CCPP	WAN	Propagation Latency	● Separating the network component ● To find patterns in the network
[60]	Clustering-based network partition algorithm: standard or improved	Simulation of network segmentation, internet2 OS3E, ChinaNet topology	CPP	WAN	Queuing latency and propagation latency	● To calculate a rough estimate of the controller population ● To reduce propagation delays between switches and controllers ● To determine the right number and place for controllers
[61]	K*-mean algorithm based	Simulations of network division, Internet2 OS3E Topology	CCPP	WAN	A controller's capacity and latency	● To determine the best number and placement that balances controller load and reduces delay.
[61]	Integer linear programming	Clustered-Based Network Virtualisation Simulations, Internet Topology Zoo	CCPP	Virtual	Average latency, worst-case latency	● Enables network operators to deploy hypervisors and controllers simultaneously in virtual SDN and dynamically add new virtual networks as needed.
[57]	The density-based switch clustering algorithm	Simulations of network partitioning, the AT&T network, and the Internet2 OS3E topology	CCPP	WAN	Time, reliability, controller capacity, and latency	● To divide the network into many similar networks ● To determine the controller's capacity restriction
[61]	Integer programming	Simulations of network partitioning, Internet2 OS3E topology, and rocket fuel topology	UCPP	WAN	Average Latency, Reliability	● Determine the ideal controller position. ● To minimise control path loss to increase SDN control dependability
[7]	Clique-based	Network Partitioning Simulations	UCPP	WAN	Link use, controller capacity, and latency	● Linking each domain to a controller ● Balancing the load on the network's controllers
[7]	Heuristic algorithms	Topology for DCP-GK and DCP-SA simulation: RF-I and RF-II	UCPP	WAN	Latency, propagation delay	● To achieve the highest level of network efficiency and precision
[61]	Controller placement Genetic algorithm	AHP technique Simulations, Internet2 topologies	CCPP & UCPP	WAN	Link usage, hop count, and latency	● To assess the performance as well as solve the location-allocation problem
[59]	Multi-controller mapping strategies	Internet Topology Zoo, simulations, and networks from AT&T and GEANT	CCPP & UCPP	WAN	Latency, capacity, Cost, and Reliability	● Attain complete resiliency ● To reduce expenditures and increase backup capacity
[57]	Two heuristic methods (Clustered and Genetic approaches)	Simulations of network partitioning, Java-based framework signal	UCPP	WAN	Latency, Reliability	● In a large-scale SDN implementation, the solution to the average-case delay measure involves ● Finding the optimal controller position

2.7. SDN technology deployments

Because large-size networks typically require a high number of controllers, the placement of these controllers has an impact on both their functionality and features [12]. CPP is a method for handling the question of the number of controllers that need to be deployed where they have

to be placed in the network, and how far apart they should be from switches [62]. According to their intended use, reducing network latency, raising network dependability, and boosting network efficiency are just a few of the categories that recent research has split placement under. The challenges of multi-controller deployment are outlined in depth in [62]. The authors give a general introduction to multi-controller systems and discuss various key issues, including network load balancing, scalability, dependability, and consistency. Along with matching remedies, they also offered some encouraging research ideas.

An analysis of an SDN-based wide-area network design was conducted in [2], addressing the challenges of controller placement in the WAN. The authors proposed the optimal location for the SDN controller based on a nondeterministic polynomial-time hard problem (NP-hard) and presented two methods for handling SDN's CPP [2]. In a similar vein, CPP was categorized in [53] according to its objectives and procedures. The study also explored the application of CPP in several cutting-edge industries, including wireless mesh, 5G, cellular and mobile networks, among others. The research underscored the importance of CPP in SDN and provided a formulation of the problem along with the underlying system model. The study in [53] offers a classification of contemporary CPP designs by objective. To reduce packet propagation time between controllers and switches, the authors of [59] introduced an innovative technique. They also presented a comprehensive examination of CPP, focusing on optimization goals such as latency, connection, cost, load, energy, QoS, and control plane overhead. The authors discussed SDN CPP approaches and suggested potential future research directions for SDN controller placement. Most literature surveys that focus on the CPP issue in SDN were also categorized by authors in [62] as segmentation and fault awareness. The report did not uncover any specific survey awareness; it merely summarized CPP techniques. Similarly, network partitioning is based on the approach using k-means, reintroduced to enhance the responsiveness of the controller and switches. In this study, the effectiveness of the k-centre and k-means techniques was examined.

A few of the CPP reviews and surveys that related to the earlier topic were highlighted. Based on their objectives and procedures, these researchers categorised CPP. They demonstrated how many authors are tackling CPP concerns by going through and articulating their challenges. On a large-scale network, SDN-based CPP continues to have issues. Reliability, load balancing, resilience, and network state consistency must all be maximised in a multi-controller SDN environment [59]. Additionally, it offers crucial information about controller deployment strategy.

2.8. Comparison of controllers' placement approaches

Since CPP entails determining the appropriate placement for controllers in terms of locations and placements, to enhance network efficiency, it has been regarded as the best method for optimising the issue when it comes to extensive SDN. The examination of the various CPP methodologies and strategies employed in the SDN is summarised in this section. Several recent research projects have been conducted to address CPP, as shown in Table 2.2. The authors in [63] used a network partition strategy for CPP as a cure for the aforenoted issue with latency. To address latency issues in a small-scale network, the Cluster-Based Network Partitioning (CNPA) strategy was proposed as an effective method for maximizing the shortest path between network nodes [63]. This strategy is beneficial as it reduces the network's energy consumption and deployment costs. The CNPA approach was utilized in [59] to ensure the shortest path for end-to-end latency between the controller and switches. Their problem formulation focused on reducing end-to-end latency and queuing delay in controllers to meet the required latency. Tests demonstrated how effectively CNPA resolved the latency problem of the CPP in a large network. The k-means algorithm was also employed to assess latency issues in WAN. In contrast to [63] and [59], who focused on using CNPA to reduce maximum latency between nodes, [62] proposed partitioning into multiple single-controller zones to facilitate load balancing between controllers and latency reduction. The work in [62] modifies the exemplar-based clustering algorithms based on affinity propagation, similar to how [63] and [59] addressed the CPP. The experiment from this study demonstrated stability and accuracy in Controller Placement (CP) in terms of reducing latency when compared to the method used in [63].

It has been demonstrated in both small and large-scale networks that the best strategy to address CPP is by adopting network partitioning. The authors of [59] prioritized scalability, privacy, and security by dividing the WAN into small domains, contrasting with [62]. In [63], the same concept was used to determine where controllers should be placed within a specific network domain to solve CPP, where partitioning was used for latency. The evaluation results of an optimization model for controller deployment and location in [57] achieved the best cost-saving in terms of controller capacity and latency reduction when compared to existing techniques [57]. One of the primary challenges with CPP is determining the number of controllers on the network. In [63], the authors discuss this problem and conclude that using the controller is still the best solution for addressing CPP as it enhances network performance. However, they also argued that the network split concept has proven to be the most successful strategy to address CPP [63], When using numerous controllers to manage CPP and address

the network performance and scalability issues [63], found that CPP was a concern. To reduce costs and improve network performance and accuracy, they varied the number of controllers in their study and dynamically assigned jobs to controllers. The issue of network performance was dealt with using an integer linear program to address CPP, virtualised SDN [60]. This model's evaluation demonstrates how latency has been optimised. The results of this study perform better than other recent placement challenges in SDN virtualisation when compared to other methods utilised to address the CPP (VSDN). Using the three important metrics of hop count, latency, and link utilisation, the model's evaluation gauges how well the process performs. This strategy attempted to assess the controller's location's optimality and then move on to controller assignment. The location-allocation problem in CPP was solved using this enhanced controller placement genetic algorithm method [61]. The authors assert that this methodology was the inaugural study to investigate how modified measurements influenced switch assignments in the CPP. The approach employed in [61] demonstrates an enhancement in resolving location and determining the optimal placement in CPP. This is in comparison to [60], which offers the location of controller modules and the quantity of active controllers for each module by amalgamating the solution of the CPP controller provisioning problem [61].
One of the objectives of this research is to determine the ideal placement for controllers' deployment in terms of their position and location to improve controller performance for fault tolerance in medium, and large-scale SDN network platforms [61]. Therefore, choosing the best approach for controller placement for the proposed SDN-FTM addressed the issue of controllers' deployment in the medium, to large-scale network experiments in this study. Therefore, a better understanding of this CPP was crucial in this study.

2.9. Tools for SDN simulation and emulation

As was covered in the previous chapter, SDN is a cutting-edge network architecture that combines the centralisation control function to make network management possible network using the controller and enabling network management using programmable capability. To realize the potential of SDN, suitable network environments are required for implementing innovative solutions. These solutions, developed by leveraging the opportunities presented by SDN, offer meaningful contributions to the future of the internet. Several simulation and emulation software tools exist in the literature that provide the necessary testing and evaluation environment for SDN. This section presents a feature-based comparison of these tools,

considering aspects such as testbed mode, emulation approach, scalability, real controller support, and many other features.

Testing the accuracy and performance of newly developed protocols and processes in networks necessitates a variety of approaches [47]. Various simulation and emulation software tools have been developed to test applications for OpenFlow-enabled networks since the evolution of the novel SDN paradigm architecture. The first approach involves performance tests on an experimental testbed, where the platform is designed using real devices. However, these are prohibitively expensive for most users and thus cannot be used. The second approach involves the use of simulation tools that model the real devices and their functionality in a software environment platform. These simulation tools are validated by most users due to their scalability, low cost, and speed compared to real-time devices. The final approach involves emulation tools, which allow physical tools and applications to be used and establish communication between simulated devices. It's important to note that these software tools have different characteristics as well as similar aspects that make them compatible and user-friendly. A brief review of a feature-based assessment of various simulation and emulation tools is presented, highlighting their challenges and shortcomings.

A. Simulation tools

A network simulator is a tool that uses a software program to design, implement, evaluate, and test the predicted network performance of a computer. Due to the complexity of communication networks, typical analytical methods cannot give a precise insight and view of system behaviour, and therefore network simulators are used. OpenFlow-enabled switches are mostly integrated into the NS-3 simulation platform, which is the new version of NS-2 and can be used to simulate OpenFlow networks, and it allows the configuration of switches via OpenFlow API. fs-SDN is created by extending the fs-simulation engine by integrating the OpenFlow-enabled controller and switches elements in it, which is one of the other simulation tools [64]. The primary goal of this simulator tool is to concentrate on the difficulty of quickly, accurately, and massively designing and assessing SDN-supported controller applications, hence facilitating a smooth transition to real-world controller software like POX and NOX [65]. EstiNet is the final one [66], this is a simulation tool for commercial networks. It is used to evaluate the events and compute the performances of controller applications and OpenFlow-enabled networks. It combines the characteristics of simulators and emulators into one package. The main idea of suing EstiNet is for a unique kernel re-entering using simulation techniques,

the simulated network's nodes can run unmodified real applications [66]. Consequently, in the case of real controllers such as NOX/POX [66] or Floodlight [66], they are capable of easily running on a host machine in an EstiNet-simulated network to control the number of simulated OpenFlow-enabled components. EstiNet produces accurate and repeatable SDN-based controller application performance results because it makes use of real-world OpenFlow network applications and the TCP/IP protocol stack.

B. Emulation tools

The main purpose of utilising these simulation and emulation tools is mostly for building and testing prototypes of networks at less cost without the need for physical devices as they may be expensive. Most of these tools were developed to simulate and emulate nodes such as switches and controllers that support OpenFlow protocol and SDN protocols efficiently [66]. For this research, we chose to study Mininet since it is the one used for simulation in this study. Mininet focuses best on the emulation of the SDN, and its main purpose is to provide affordable fast and simplicity for building prototype models as well as evaluating SDN protocols and other related applications [67]. Mininet as the software makes use of OpenFlow Software-Based switches to visualise network models and offers correct corresponding hardware-based OpenFlow switches' semantics. It follows that when you design, implement, and test controller applications in Mininet, the same architecture can also be deployed in a real OpenFlow-enabled network in real-time with no need for any modification. Being an open-source type of software tool, Mininet has various build components that can be utilised in a variety of configurations to suit the demands of users.

Mininet-HiFi is a development aimed at enhancing container-based emulation. This environment, constructed from switches, network links, and virtual host machines, operates by integrating authentic application and kernel code with a software-emulated network. It runs on modern multi-core server components atop Mininet [8]. Mininet-HiFi incorporates mechanisms to carry out resource provisioning, performance isolation, and system performance monitoring, leveraging lightweight, OS-level virtualisation techniques. A key objective of Mininet-HiFi is to augment the reproducibility of networking research. For wireless SDN environments, Mininet Wi-Fi [68] serves as a solution that facilitates experiments emulating real-time Wi-Fi networking scenarios. It is an extension of the well-known emulator, Mininet, and includes additional virtual wireless stations and access points integrated into the emulation platform. This is achieved using the Linux wireless device driver,

without sacrificing SDN capabilities or the lightweight wireless visualisation software architecture. Mininet CE [8] and SDN Cloud DC, as implemented by [66], are expansions of Mininet designed to support simulations of large-scale network architectures. The standard limitation of Mininet, which supports up to 2000 nodes in the emulated network, renders a unique form of Mininet less applicable in a global network context. To address this limitation, Mininet-CE introduces higher-level software atop Mininet. This amalgamates multiple individual instances of Mininet into a single cluster, which can be either a virtual or physical machine running the Linux OS. This approach effectively extends the scalability of Mininet, making it more suitable for larger network simulations.

Similarly, SDN Cloud DC [66] is a novel framework allowing the implementation and evaluation of novel OpenFlow-supported controllers for cloud data centres. This framework is developed by augmenting the incorporation of both Mininet and POX controllers with all the software modules essential to emulate novel management and control of network resources such as (SDN-based intra-DC network) network policies, network traffic, DC topology discovery, and many others in the SDN-based intra-DC network. According to the literature, the emulation environment based on distributed methods has been suggested to support large-scale experiments. Mininet is one of the platforms that support; large-scale experiments; it is developed to increase Mininet to widen an emulated network over several physical machines or nodes. This technique was developed to make available quite a few thousand nodes to emulate, as well as permits large-scale forms of network experiments. Among other approaches is the Distributed OpenFlow Testbed (DOT), which also permits the provision of network emulation through a cluster of physical or virtual machines [67]. The Distributed OpenFlow Testbed environment provides the best network and computation power for switches and other mimicked components, links, and hosts, unlike Mininet and therefore can merely grow with network size and traffic volume to replicate aspects such as huge data centres and wide area networks (WAN). MATLAB also provides a virtual model to simulate and test computer systems [69]. Most of these models are designed and validated with physical models before being tested as a prototype. Making use of an SDN radio combined with MATLAB and Simulink desktop environment for design processes, and simulation of the system can also be used to use to analyse cluster property using the smallest distance between data points as a metric [69]. This method is primarily used to determine the number of controllers that minimises the overall network propagation delay for switch-to-switch latency within the context of SDN controller placement, as well as the ideal placements for the controllers.

Several algorithms can be used to address these issues, but for this study, we employ the city-block distance metric in MATLAB to determine the number of controllers that must be deployed and their precise placement throughout the network using k-means and k-medoid clustering [69]. For this study, Mininet was chosen as the suitable emulation to design, evaluate, implement, and test the performance of the SDN-FTF. Table 2.3 below shows a comprehensive outline of the comparison of the simulation feature-based and emulation tools for SDN discussed in this chapter.

Table 2.3: Feature-based comparison of emulation and simulation tools [115]

Name of the software	**Testbed mode**	**Emulation approach**	**Programming language**	**OpenFlow version**	**Size of network scalability**	**Real controller support**	**GUI support**
Mininet [70]	Emulation	Centralised	Python	1.3	Middle size (1 to 200 Nodes)	Yes	Yes
Mininet-WiFi [71]	Emulation	Centralised	Python	1.3	Large (Wireless Network)	Yes	Yes
Mininet-CE [70]	Emulation	Centralised (multiple Mininet instances)	Python	1.3	Large	Yes	Yes
Mininet-HiFi [70]	Emulation	Centralised	Python	1.3	Large	Yes	Yes
MaxiNet [72]	Emulation	Distributed (multiple virtual machines)	Python	1.3	Large	Yes	Yes
Estinet [73]	Emulation & simulation	Centralised	C/C++	1.3	Large (Wireless Network)	Yes	Yes
SDN Cloud DC [54]	Emulation	Centralised	Python	1.3	Large	Yes	Yes
DOT so [68]	Emulation	Distributed (multiple virtual machines)	C++	1.3	Large (WAN, DC network)	Yes	Yes
f-sdn [74]	Simulation	---	Python	1.3	Large	Yes	Yes
ns-3 [75]	Simulation	---	C++	0.8.9	Large	Yes	Yes
MATLAB [74]	Emulation & simulation	Distributed, data acquisition, (multiple virtual machines	Python, c, C++, PLC, Verilog, and VHDL Code	5.3.1	Large (WAN, wireless, SDN placement)	Yes	Yes

2.10. Summary

This chapter presented a list of articles and related works in terms of SDN. It starts by introducing SDN as an emerging cutting-edge networking model that addresses the fundamental problem and network complexity. An overview of traditional networks and the benefits of SDN were discussed, it was argued that SDN guarantees the scalability, dependability, and availability of the network, both physically and intellectually in a single centralised and distributed system. The issues of architecture for cloud-based on-demand services were discussed over access control and service verification and SDN architecture was proposed in the literature to provide appropriate protocols for forwarding decisions for cloud-based on-demand services. Furthermore, it explains the concept of SDN-fault tolerance and its importance in SDN platform networks. The chapter also provides various explanations for the SDN-fault tolerant mechanism that existed that are used to address consistency in both controllers and switches in distributed systems and gives some challenges as well as classifications of various SDN-Fault Tolerant. In addition, it explains some issues related to the implementation of the SDN-fault tolerance as well as SDN controller placement and its significance in the network's SDN controller deployment. Among other unsolved problems and persistent issues with SDN-FT, including controller placement and Scalability and load balancing, network partitioning, real-time fault detection, security and resilience, and dynamic topologies were also presented. In conclusion, a theoretical analysis of controller placement schemes and different simulation and emulation tools suitable for implementing SDN were discussed.

Chapter 3

Research Methodology and Design

3.1 Introduction

In Chapter 2, different scholars and researchers have proposed various definitions of research depending on their fields of study. According to the Oxford Dictionary of English (1986:720), research is a systematic investigation process used to either discover new facts or supplement information to increase or improve current knowledge. According to [70], research is "delight and systematic inquiry or investigation in an area, to discover or revise facts, theories, and applications". From the above definitions, scholars such as [76] [71], have argued that the definition of research should be provided and elaborated on within the context of a given research in a specific field and related branch of science. As a result, they offered a definition of research, which they define as "multiple, systematic strategies to generate knowledge about human behaviour, human experience, and human environments in which thinking and action process of research are specified so that they are logical, understandable, confirmable, and useful" in their field of study, which is humanities [70]. In contrast to hard science, [76] [71] redefines research as an organised, methodical, and logical process of inquiry that uses empirical data to test statements that can be true or false the researcher can use to answer a question. Following the above definitions, it can be summarised that research is a systematic plan of activities that have the objective of establishing novel facts and information about a specific phenomenon of interest. Therefore, it is crucial to apply the research method to this phenomenon of interest, which includes identifying the problem, gathering data, analysing the data, and reporting the research's conclusion.

Computer science (CS) is a discipline that studies systematically, the different algorithmic processes that describe, and transform information in theory, design, analysis, implementation, simulation, and application [70]. Dodig-Crnkovic [77], assert that CS is the study of computer-related phenomena while authors in [70] viewed it as research on the mechanisation of abstraction. This supports the idea put forth by [76] that the main goals of CS research are theory and abstraction since it uses the computer as a problem-solving tool rather than as a science. Several scholars have also argued that CS does not have the aspect of the knowledge field as a science [77] [76]. The theoretical concepts' study, practical disciplines in the development and implementation of computers for data processing, the resolution of

mathematical and logical issues, and many issues in other scientific fields can be characterised as the field of CS. Therefore, CS research is very important due to the dynamic nature of the discipline. Research is thus, used to ensure that the CS activities do not become obsolete by always finding a new way to improve existing problem-solving approaches or develop new efficient approaches.

Additionally, to carry out research in CS, there are numerous processes, algorithms, and methods that are used in a research investigation, commonly known as research methodologies. That is, any form of research must include crucial components called research methodologies which provide an appropriate procedure to conduct the research from the beginning to the end. Moreover, to effectively answer the defined research questions, research methods are numerous designed procedures, such as theoretical, experimental, design, numerical, and evaluation, which aid the researcher in gathering data, creating samples, and producing solutions to specific problems, according to the literature. To produce a solution, a research process needs a variety of equipment and tools. Authors of [76] [77] assert that in theoretical research methodologies, the researcher must present evidence to back up claims based on already-known facts. The research must, however, describe the facts that have been gathered, measure them, and then observe them through scientific research procedures [76].

Therefore, in the context of this research, different research methods in CS research were employed to answer the research questions defined in Chapter 1 section 1.5. We explained the concept and technique used to design and evaluate the proposed SDN fault tolerance model and the ODSS in detail. Accordingly, DSR was adopted as the suitable research methodology for this study. Although DSR was only briefly introduced in Chapter 1, a more thorough explanation is provided in the next section. Figure 3.1 presents the workflow of this research. As shown in Figure 3.1, the research started with a preliminary literature review to identify relevant and practical research problems to research on followed by a comprehensive literature study to gain a broader understanding of the problem and related areas, technologies, and tools. This led to the writing and defending of the research proposal which was presented in the department and submitted for approval alongside ethical clearance. Moreover, based on the literature study carried out, the knowledge gained led to the selection of the appropriate research methodology and methods for answering the formulated research questions. Accordingly, the research was designed and followed strictly in terms of execution. Appropriate steps were taken to evaluate and validate the results obtained to increase its trustworthiness. Lastly, the actual research was concluded, written and submitted to the institution for external evaluation.

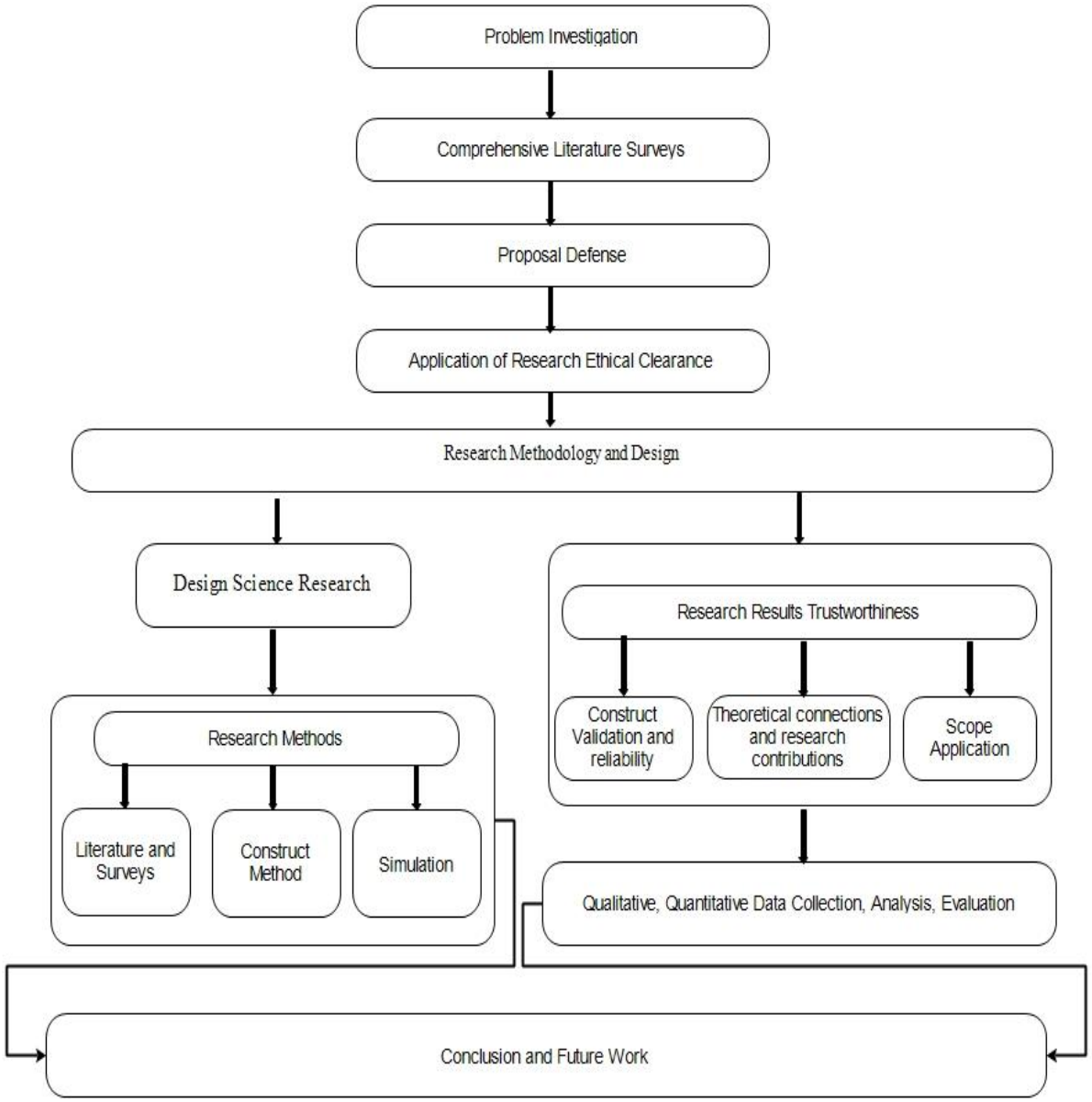


Figure: 3.1: The research workflow

3.1.1 Outline

This Chapter presents the different paradigms, research methodology and research methods employed in this research. It started with a brief introduction followed by methodology and the associated paradigms. The Chapter also discusses various research methods and gives their application within the context of this study. Furthermore, the data collection and analysis process were also explained as well as a research evaluation process, reliability, and validity that were considered when conducting this study, together with the administrative procedures and ethical considerations that were taken into consideration.

3.2 Research methodology

Science uses a methodical approach called research methodology to explore how research is conducted to find solutions to issues. The researcher's responsibility in this procedure is to describe and clarify the occurrences that the study predicts. According to the literature, research methodology is a procedure in which the researcher outlines, justifies, and foresaw the various tasks that would be carried out throughout the investigation. Authors in [70], define research methodology as a strategy for employing theory to elucidate the research procedure. According to some researchers, research methodology elaborates and offers a detailed description of the sort of problem that must be examined, what defines the research challenge, and how to select the best design and methodologies to produce the results [70] [76] [77]. It is a process that includes the key premises for the research issue, the foundational ideas that must be established, and the steps by which various investigation methods can be applied. To put it another way, research methodology tries to define the approaches that can be employed to gather knowledge while also highlighting the defined research scope. In addition, research can employ techniques, algorithms, theoretical, statistical, experimental, and simulation as part of a research study to identify a solution to a particular problem. Still, the researcher must create a workable methodology for the research topic and ensure that the right procedures are being applied. When conducting research, a variety of approaches and research paradigms can be used. The research paradigms employed in this study as well as others are discussed in the section that follows.

3.2.1 Research paradigm

A paradigm for research is a way of thinking that the researcher uses to unearth new, accurate knowledge about the study object [78]. A paradigm is a group of presumptions, assertions, or concepts that provide a framework for inquiry and reasoning [79]. In other words, it acts as the basis for the research, which entails choosing a research method, formulating the problem, and gathering, processing, and analysing the data [79]. The following section discusses various research paradigms.

- ***Interpretivism***: This approach seeks to explain social interaction by using study methods that place a preference for ideas, reasons, and reasoning above facts and figures. According to interpretivism, social creations, interpretivism, consciousness, shared meanings, and tools are how we gain access to reality [78].

- ***Positivism***: The idea that reality exists independently of individuals [78]. To operate freely, the socialist adopts the role of an objective analyst by distancing himself from personal ideas [76].
- ***Pragmatism***: According to this method, the importance and significance of choices and facts found in research data are evaluated in light of their real-world implications, giving them justifiable acceptability [76].

These two paradigms can accept many of the research paradigms applied in the field of information systems (IS) research. However, according to the [73] perspective, they fall short of fully covering DSR. A third paradigm that coexists with positivism and interpretivism was suggested [80]. This paradigm, also known as critical realism pragmatism or developmental paradigm, can fill in the gaps where DSR's theory falls short. In most other situations, technology and software development are viewed as values that may or not be present. In this paradigm, the development and testing phases of technology and software bear a significant amount of responsibility. As a result, when this software is absent, some significant organisational problems cannot be resolved. The pragmatic perspective is concerned with how technology is created and how it might positively influence both individual and organisational experience. The IS environment can also be thought of as a socially implemented system. Because of its developmental nature, this paradigm is analogous to DSR, according to [80].
The DSR technique employed in this investigation was the main point of emphasis within the context of this study. According to the literature, the DSR approach is a framework for problem-solving research that attempts to develop artefacts that can be a method, product, methodology, procedure, a combination of any of these, or any other means of attaining a goal [75]. DSR methodologies have been utilised in a variety of fields, including engineering, software engineering, architecture, information systems, and computer science. However, how research design is used varies slightly depending on the discipline mentioned above. Therefore, in this thesis, DSR was identified as the most suitable and relevant methodology to be used to tie the problem of this project and its solution together with accumulated theoretical knowledge gained from the literature and various existing systems.

3.2.1 Design science research

DSR is a relatively new type of method of study to create, instead of attempting to explain an existing reality, create a new one [79]. The DSR was described as having a dual mandate firstly, to acquire knowledge that is used to address issues, bring about change, or enhance current

solutions, and secondly, to produce new information, theoretical explanations, and insights. Considering how more pertinent could help close the gap between theory and practice research that yields recommendations beyond descriptions, explanations, and predictions is encouraged, according to [79]. This type of study, which adopts prescriptive mythology, discovers assistance in DSR.

According to [79], DSR is a research methodology that places a strong emphasis on developing studies with a focus on prescription, project, and artefact construction. As a result, it is a technique that is interested in potential solutions as well as the problem itself [79]. There is a study based on the design science paradigm, which, according to some researchers [74], seeks to design artefacts and suggest fixes to present issues to enhance or create new systems. However, there is also research based on traditional sciences, which investigates complex natural or societal phenomena to explore, describe, explain, and, if possible, predict them [81] [74]. Several studies have shown that DSR seems to be a responsible method that strives to permit the creation or creation of artefacts, or even to prescribe a solution, as well as operationalise research [81], creates a systematic process for designing and developing artefacts that can answer various challenges by facilitating the development of research for numerous distinct sectors. Another distinguishing factor of DSR in the literature is that despite being problem-solving focused, it focuses more on finding a workable solution to the challenges at hand than an ideal on [81]. Numerous researchers have made recommendations in the literature, see Figure 3.2 of adapted processes that can be followed when using DSR [82]. Figure 3.2 shows how DSR is driven by the desire to enhance the environment through the introduction of fresh and inventive artefacts as well as their construction methods [83]. The scientific information that can be produced through creating an artefact can be used to describe the DSR philosophy [83]. Additionally, according to Figure 3.2, a fresh and inventive artefact must be created as a solution to a well-defined and pressing business issue, which scholars have referred to as problem awareness [82]. Accordingly, problem awareness is the recognition that there is a specific issue that has been identified, and one can begin to research it using the literature that is currently available. A potential design solution can be presented as an artefact through literature research [84].

The exploration of perspective topics and ideas for fixing the problem through a literature review is finished once problem awareness is in place, and proposals regarding a possible suggested solution are in the form of an artefact [84]. To assess the calibre and effectiveness of the potential artefact, clear objectives and limitations must be set [84]. The research's output should advance the field, and the artefact should be developed using appropriate research

techniques [84] The design process is often iterative, requiring multiple revisions to adjust the artefact to the original specifications [84]. Planning for various iterations is essential when developing a study design. It should incorporate more than just trial and error, be theoretically sound, and adhere to standard data collection and processing techniques [84].

3.3 Research design and methods

The research methodology and methods utilised to address the research question are presented in this section.

3.3.1 The stages of the research study

This section gives an overview of the procedures used in this research investigation. The research started with a thorough examination of the relevant literature to establish a strong foundation for the application of the SDN FTM and ODSS. The review includes keywords from publications in journals and other pertinent documents that range from 2008 to 2022. The topics being tackled in this study have been the subject of methodical data gathering to discover both practical issues and theoretical answers. After having a detailed understanding of secondary data, an SDN fault-tolerant framework and security service on-demand system were designed and tested using quantitative and qualitative methods. Furthermore, data were analysed by evaluating the system during simulation, then trustworthiness was demonstrated from the results. Figure 3.2, shows a summary of the steps that were used in the investigation.

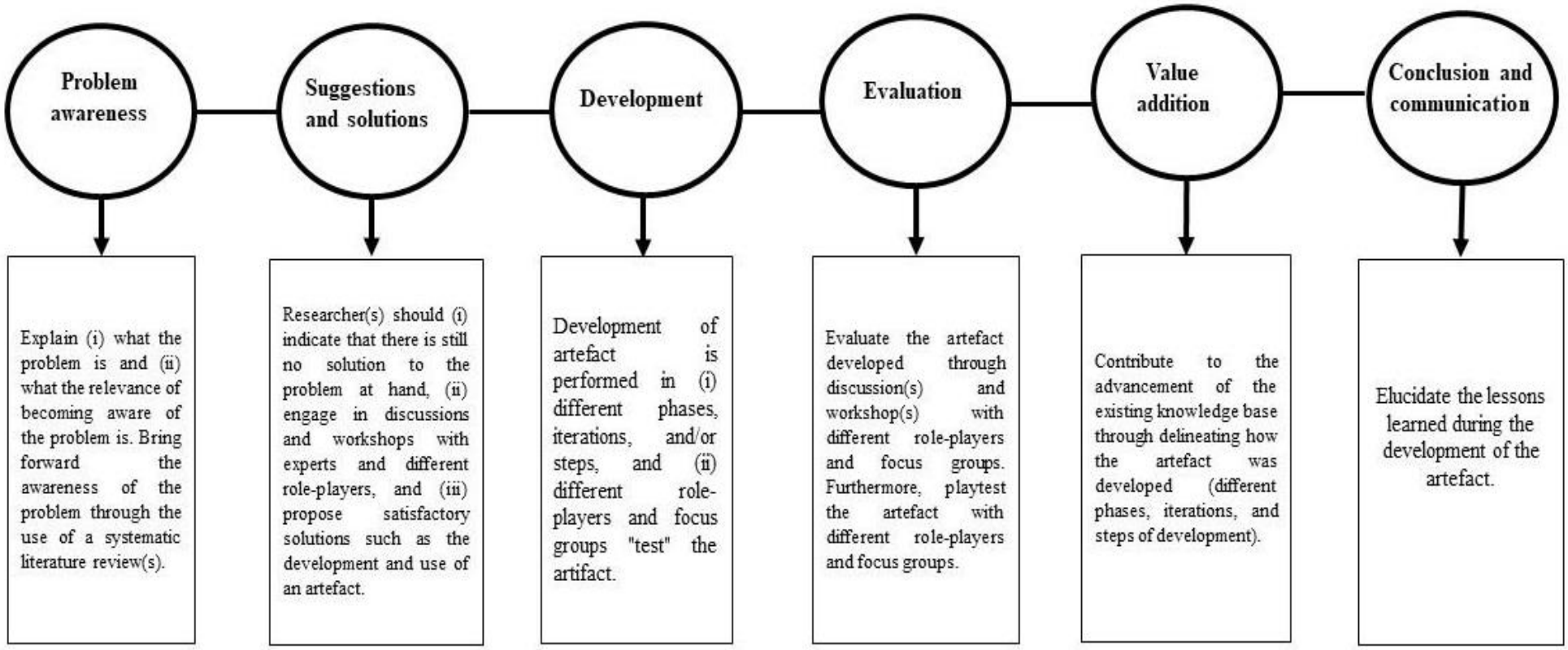


Figure 3.2: Summarised design science research process with guidelines and description [84].

3.3.2 Design science research applied in this study

According to Figure 3.2 in the previous section, the design science research process often entails various steps or activities, including the construction of artefacts, to choose the ideal answer to the Classified issue [85] [86]. Figure 3.2 was used for this research study in the form of the interactions and steps. Below are the three iterations that were carried out throughout design science research, the procedures that were taken during each iteration, and the research questions and objectives for each iteration. Additionally, a combination of Figure 3.4 is offered as an overview of the DSR procedure used for the development of a fault tolerance model for the SDN controller as well as ODSS based on SDN functionalities.

The DSR is made up of multiple iterations or design processes. The layout and development of the suggested fault tolerance framework and the ODSS were created using a mixture of the DSR processes shown in Figure 3.4 [86]. The next sections provide detailed explanations of the number of iterations and procedures used to design and develop the SDN controller fault-tolerance framework and SDN-Based ODSS.

1) Iteration 1 and step 1: Problem Awareness

In the literature, researchers in DSR have recommended that academics using DSR carry out end-user research to gather insights as well as learn about the users' active and latent wants and values. Therefore, academics that employ DSR in their research, studies, and projects ought to be aware of what users feel, why they act in a certain way or do not act at all, what to do, how they feel about the issue, the cultural, political, legal, and social circumstances as well as how they think. The procedure is viewed as problem awareness, considering Figure 3.3. This study specifically aimed to determine whether the proposed models can be a suitable system for addressing fault tolerance issues as well as offering the best on-demand security service for users. This technique is explained in Chapter 3 using the various results from the literature studies that were done as well as the context and content analysis.

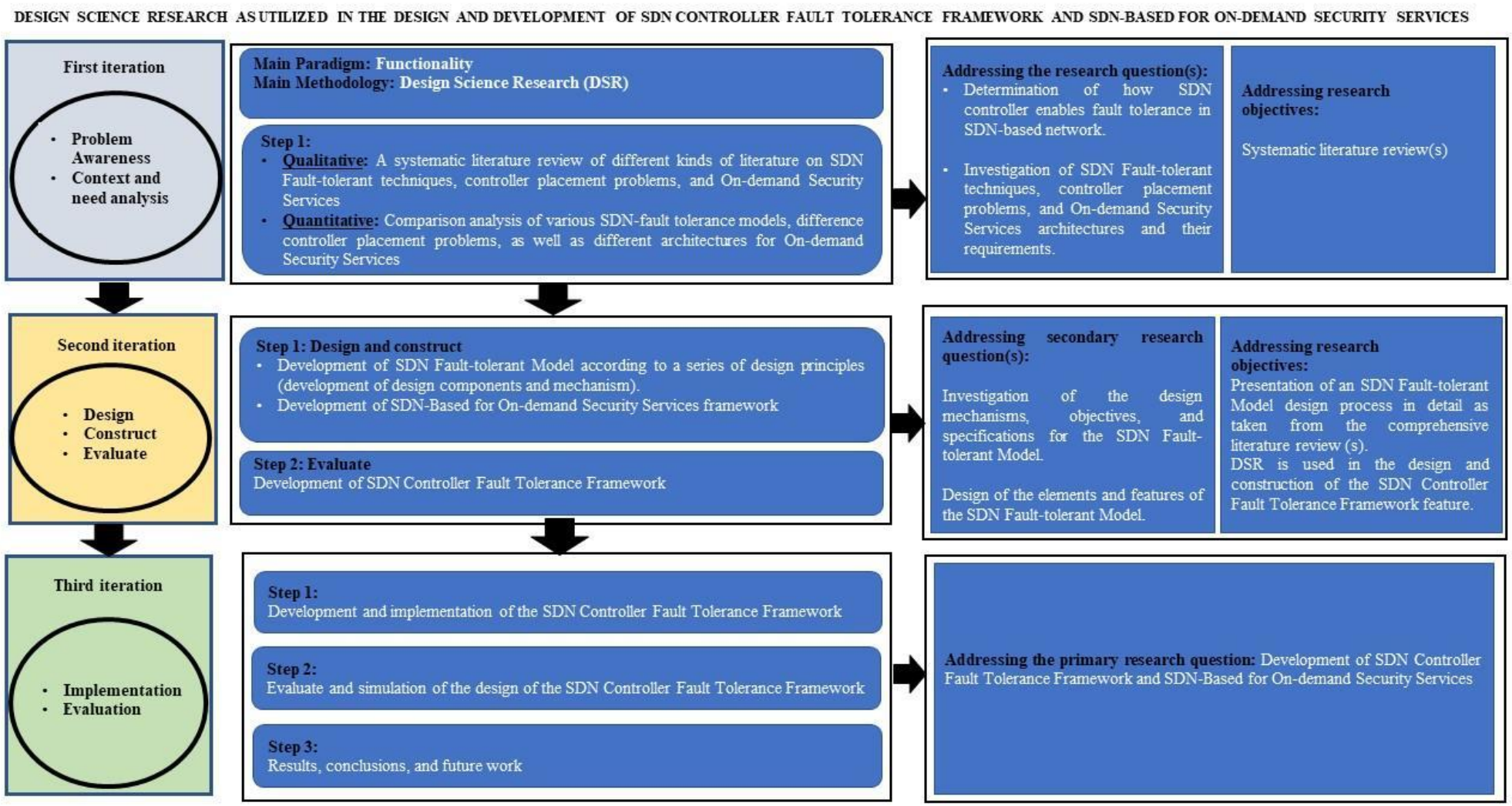


Figure 3.3: The design science research process

2) Iteration 2 and step 1: Design and construct

DSR enables the use of many methodologies (such as feature trees, experience sketching, and many others) to collect various ideas for resolving a user problem and ultimately arriving at a solution. The SDN fault-tolerance techniques, CPPs, and on-demand security services were developed in this study, as shown in Figure 3.4, using a series of design principles (development of design components and mechanism). Chapters 4, and 7, describe this procedure in detail.

3) Iteration 2 and step 2: Evaluate through testing.

DSR advises testing the ideas, solutions, or artefacts to ascertain the best solutions. When the proposed SDN fault-tolerance models and the SDN-based on-demand Security Services framework were being developed, this procedure looked at different designs in the literature and added missing gaps in them, and therefore the new design was developed. The input and investigation are the added new components in the proposed framework. Chapters 4, 5, and 6 explain this procedure.

4) Iteration 3 and step 1: Implementation

One of the methods referred to in Figure 3.4 which allowed the improvement of the design components, features, and characteristics was reviewing, improving, and testing the solution to acquire the best one that can be generalised. The components of the system were able to be created through these techniques, including in the simulation. Chapters 5 and 6 explain this procedure.

5) Iteration 3 and step 2: Evaluation

The proposed SDN fault-tolerance model's framework was evaluated employing various scenarios using simulations to validate all components of the design functionality and reliability, Chapters 5 and 6 explain and demonstrate the procedure.

6) Iteration 3 and step 3: Results, conclusion, and future work

According to DSR, the theories that already exist can be compared to the developed solution or designed artefacts, and the learned information can be distributed to already current DSR communities. Additionally, it is feasible to develop current stories further. By publishing articles, this study evaluated the proposed frameworks. The DSR procedure is described in detail in Chapter 8.

The DSR framework utilised in this study is combined with the objectives of this study and is outlined in detail in Figure 3.4.

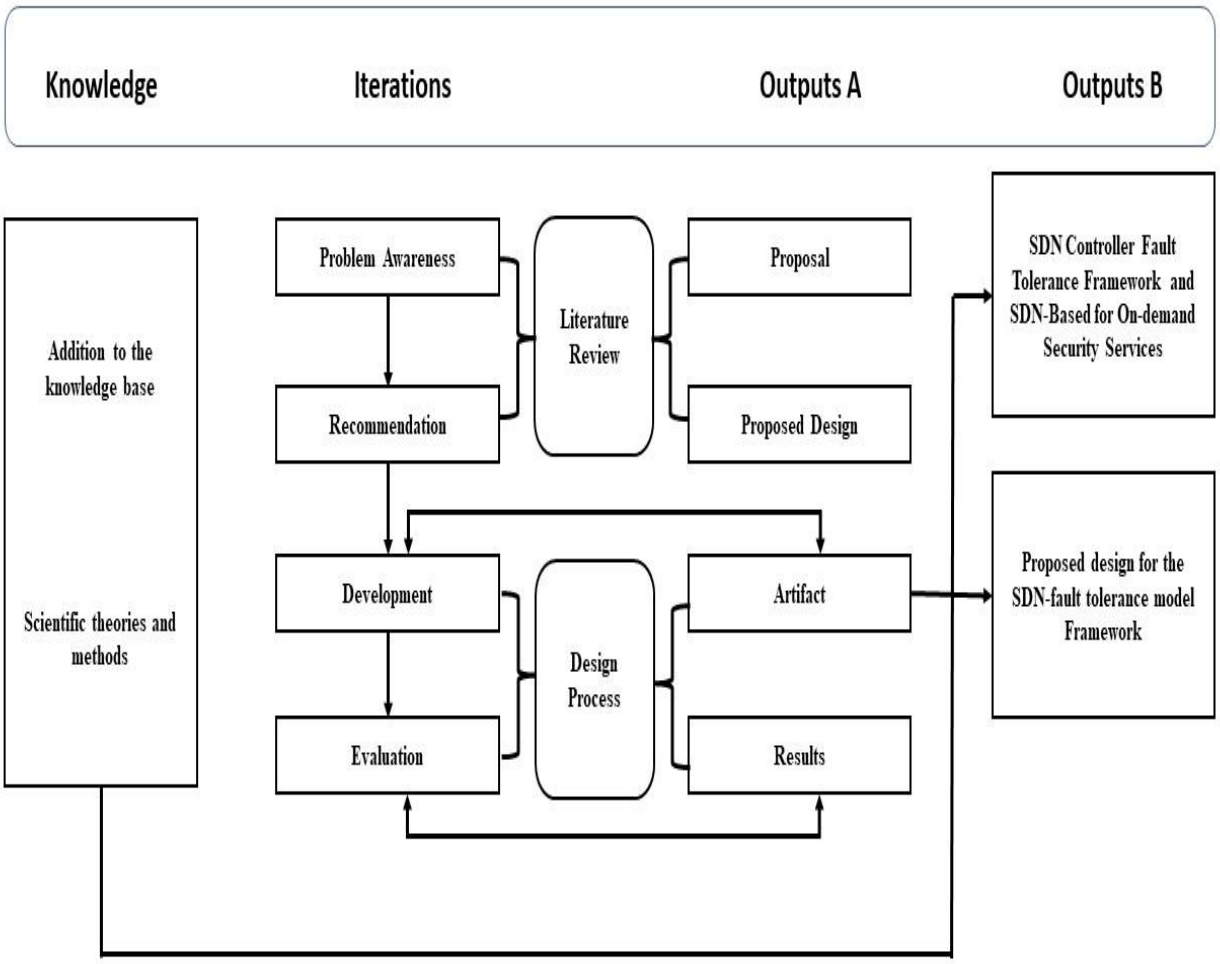


Figure: 3.4. A summary combination of the DSR process with the research objectives of this research adapted from [87] and [88].

3.3.3 Research methods

According to [88], research methods are the strategies and procedures for a study that shift the focus from general hypotheses to procedures for data collection and analysis. Theoretical suppositions, strategies, and exact techniques are all combined. The research problem can be approached methodically using research methodology. In research methodology, we talk about the research methods and the justifications for each strategy in the context of the research project. To evaluate the research findings, we justified the choice of a certain approach or technique over others.

Both qualitative and quantitative methodologies were employed in this study, with consideration given to the pragmatic and the nature of the problem addressed. The qualitative approach provides the researcher with a means of understanding a phenomenon by observing or engaging with the study subjects [88]. For this reason, phenomena that emerge in the framework of the study are of interest to qualitative researchers to be investigated and/or understood. The quantitative approach provides a research strategy that explains a phenomenon by collecting numerical data and analysing those data using statistical techniques [89]. It is a methodology in which the researcher acquires information utilising certain devices that give statistical data while also using investigation approaches like surveys and experiments [89].
The following research techniques were used in the context of this study to address the research topics listed in Chapter 1:

A. *Literature review/survey:* This is the core qualitative aspect of this research where several research materials were reviewed/surveyed to identify and understand the problem and to find different solution approaches to apply. The goal here was to broaden the researcher's knowledge. This was achieved in Chapters 1, 2, 3, 4, 5, 6, and 7. To achieve this, we studied several academic studies and technical reports on fault-tolerance management, fault-tolerance techniques, SDN architecture, and all the issues around them. To build a strong foundation for an argument supporting the adoption of the SDN fault-tolerance framework and security service on-demand includes keywords from publications in journals, conference proceedings, and report materials that range from 2008 – 2022. Furthermore, data analysis was done quantitatively by evaluating the simulation system to demonstrate the trustworthiness of the results. Additionally, various network simulators were investigated. An in-depth review of the literature and surveys was implemented in Chapter 2 whereby clear discussions and explanations of various works in the literature were applied to respond to both the primary and secondary research question 1.1 as shown in Table 1.1. Moreover, we compared existing ODSS frameworks to design the architecture presented in Chapter 7.

B. *Model design:* This is a constructive method that relies on a variety of research tools and is a problem-based approach to solutions. Its goal is to produce original answers to issues that are both practically and conceptually pressing. According to academics [97], operational research practical approaches based on both the hard and soft paradigms might benefit from problem-solving and problem-construction methodologies. Building models, systems, algorithms, and diagrams to solve problems is essential in

the practical and applied field of computer science, which includes software engineering and project management. Therefore, this method was applied to Chapters 3, 4 and 7 to identify the suitable research methodology and the design and the two frameworks proposed based on the information gathered during the literature review/survey, this study's secondary research questions 1.2 and 1.3 were addressed, we designed the proposed controller fault tolerance (Chapter 4) and ODSS frameworks (Chapter 7) and discussed their functionalities.

C. *Simulations:* The simulation method, as defined by [44], is the use of a logical or mathematical model as a research method to address issues with a referent system. In computing research, simulations are employed, which require the development of a fictitious environment by the researcher using specialised software to conduct testing [44]. Therefore, this research applied simulations to measure the effectiveness of the proposed FTM designed in Chapters 5 and 6. The Mininet [8], simulator tool was used in the evaluation to answer the secondary research question 1.4.

3.4 Data collection and analysis

Setting the parameters of the study, gathering data through unstructured or semi-structured observation and interviews, documents, and visual materials, as well as defining the protocols for capturing information, are all steps taken during the data collection process [44]. The following methods were employed in this research to collect and evaluate the research data:

3.4.1 Data collection

The researchers made use of primary and secondary data collection when conducting this research. Primary data was collected during various steps of the simulations. Various components of the systems were tested using different scenarios to collect data that was later analysed. The researcher used secondary data by gathering journal publications, conference proceedings, students' work, and reports materials as shown in all the chapters.

3.4.2 Data analysis

As discussed above, the primary data was collected during various steps of the simulation process as indicated in the previous sections and analysed using various tools. The controllers were tested using the design topology to evaluate various parameters to address the research problem. Data were analysed using MS Excel to allow the depiction of a great deal of information using defined variables that were described, and results were presented utilising visualisations such as bar charts and tables to be digestible for all types of audiences using

descriptive analysis. The literature reviews were conducted as part of the secondary data and analysed using various comparative analysis techniques whereby numerous concepts of this study were compared among them to find out the best approaches in the development and evaluation of the proposed FTM.

3.5 Research evaluation, reliability, and validity

This section presents the steps taken to ensure the designs and results presented in this research are reliable and valid.

3.5.1. Evaluation

After identifying a suitable tool for performing experiments, a couple of steps were developed to evaluate the system using different metrics and parameters such as system properties that define the types of hardware and software that were used during the evaluation and their respective values. Furthermore, evaluation metrics were defined to specify each metric used and their description, distance metric as well as simulation parameters and node details.

3.5.2. Reliability

To ensure the reliability of the system, we made use of test-retest reliability whereby, the system was tested repeatedly using the same parameters and evaluation metrics to check its performance and consistency. The results of the system's test were compared with other similar frameworks and in terms of performance looking at the results and parameters used during the testing showed that the system is reliable.

3.5.3. Validity

The validity of the proposed framework was based on two measurements that were used as types of evidence, construct validity and content validity. To ensure validity, the proposed system was compared with other existing frameworks using various theories, design principles as well and knowledge gained in the literature. The system's performance in comparison to other similar systems and literature indicates high construct validity. To ensure the content validity of the proposed system, all components of the system were evaluated, and their functionality was tested to ensure that they covered all aspects of the system that was evaluated.

3.6 Administrative procedures and ethical consideration

In compliance with an ethical procurement protocol, this study received ethics clearance from North-West University, Mafikeng Campus. Following the successful defence and approval of the proposal, the researcher submitted an ethical clearance proposal to the faculty's research ethics committee. This committee, an internal faculty body, reviewed the application. Upon their review, they issued an ethics clearance confirmation with the reference NWU-01590-20-A9. This clearance, granted by the Faculty of Natural and Agricultural Sciences Ethics Committee (FNASREC), is valid for the duration of this study.

3.7. Summary

The research design and technique that underpin this study are described in this chapter. Various definitions of research depend on their fields of study in the existing literature, from different scholars and researchers were discussed and explained. For this study, design science research (DSR) was adopted as the suitable research methodology in CS research to answer the research questions in this study. To do that, the concepts and techniques used in this research to design and evaluate the proposed SDN fault tolerance model and the ODSS were elaborated on in detail. Using design science research, the workflow of this research study was presented more explicitly, and multiple DSR iterations and design processes were used as ways of investigation to achieve the objectives of this study. In addition, various processes were used as ways to collect data and analyse them to ensure the reliability and validity of the study.

Chapter 4

Controller Fault Tolerance Model

4.1 Introduction

The OpenFlow-based SDN design uses a single controller which makes the system vulnerable to a single point of failure [90]. Failure of the control plane is crucially important in SDN since a faulty controller can disrupt the entire network. In recent years, multiple-controller architecture has been used to overcome the challenges presented by the single controller by ensuring network scalability, flexibility, reliability, and availability. To overcome a single point of failure, there is a need to put in place a vigorous fault-tolerant scheme that ensures the network's continuous functionality in any event of fault or failure.

Therefore, this chapter presents the controller fault tolerance model as a viable approach to address failure on the SDN platform to achieve reliability and availability of the network and its services. The goal of the proposed model is to address the issue of fault tolerance on small to medium-sized networks to optimise their operation and make them scalable and ready without any cost on network resources. The section that follows presents the system model and explains in detail all its components and functionality [26].

4.1.1 Outline

This chapter presents the proposed SDN fault-tolerance model (FTM) for SDN. It begins with a brief system overview, assumptions to ensure that the designed system is fault-tolerant, and the FTM design with different components and their algorithms. The chapter concludes with the overall operation of the designed SDN-FTM [26] [90].

4.2 Assumptions and design principles

This section discusses the various aspects strictly followed in the design of the FTM. In essence, it presents the assumptions taken to ensure that the design system is fault tolerant; different design principles that lead to the choice of the FTM as well as the requirements of the proposal system.

4.2.1 Assumptions

Within the context of this research, the FT design's goal is centred on achieving reliability, availability, consistency as well and optimum placement of the controllers, which are the main features that ensure the dependability of the SDN platform. To ensure that the proposed FTM is fault-tolerant, the following assumptions are being considered in the FTM design:

- The focus is on small to medium-scale SDN networks. It can also be improved for large scale where needed.
- The proposed FTM is made up of a set of events that defines its entire operation.
- In the case one controller fails or develops a defect, the network shouldn't be affected, and the remaining controllers should continue to function.
- To ensure that data is not lost, the proposed FTM uses the state replication method whereby information among the controllers is shared in real-time of any changes on the network.
- The proposed FTM uses a port statistic reply request technique to reduce the controller's burden or to prevent overloading the controller with more events.
- Multiple controllers integrated with a stationary agent are deployed and operate independently from one another.
- To reduce the controller's burden or pressure, an external stable database was used to store all the information about the network.
- To ensure optimised placement, the K-centre algorithm is used to define the number of controllers required and the controller workload.

4.2.2 Functions

The proposed FTM is made up of four controllers that operate independently from one another, a stationary agent (SA), and a stable database. However, this design can also be used in a multi-controller architecture to carry out failover procedures that may cause the network to malfunction. The system function includes:

1. The proposed FTM shall monitor the network and the SDN controllers in real-time and detect faulty components.
2. Once a fault is detected, the proposed FTM shall implement the FT strategy and recover from the failure within 40 seconds.
3. The proposed FTM shall apply a synchronisation strategy using active replication to ensure that failure does not affect the operations of the network. Active replication

ensured real-time synchronisation of network states to ensure how readily accessible the network was.

4. The proposed FTM shall choose a replacement controller to assume the duties of the primary controller in the event of a defect or failure of the primary controller. A new controller was chosen based on information from the OpenFlow network port statistics regarding things like load balancing and latency.
5. The assumption of the design for using the stable database out of the controller like other designs in the literature [91] [92] was to avoid the issue of overloading the controller with other services that would affect its performance [92].

4.2.3 Design principles

This section covers the design guidelines for developing a fault-tolerant SDN control plane. To decrease the possibility of a single point of failure and other issues brought on by centralised SDN architectures, the design of the suggested FTM in this research makes use of a distributed SDN controller's architecture. Furthermore, based on the actual layout of the controllers, a flat design is guaranteed a global view is adopted which assumes an active replication strategy [11] [6]. The choice of this design originates from [93], with the idea of meeting the response time requirements of the network during the implementation, which is simply reducing the average controller latency, improving the controller plane's dependability, and guaranteeing the network's availability. This is expressed in equation 4.1.

$$Availability = Min\ (Avg\ latency) + Max\ (Rbty) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.1)$$ [93]

In this research, the proposed FTM uses a design depending on the nature of the operation, and active replication strategy and requires minimal modification to the original SDN controller by introducing a stationary agent (SA) as shown in Figure 4.1. The function of the SA is to ensure effective monitoring and communication between controllers to achieve a robust fault-tolerant system. The active replication strategy ensures that all controllers maintain an active connection with the data plane, but only one controller (primary) takes charge of the network operation at a time. In the case that the primary controller fails, the secondary controller is chosen to serve in the primary position depending on load balancing and latency.

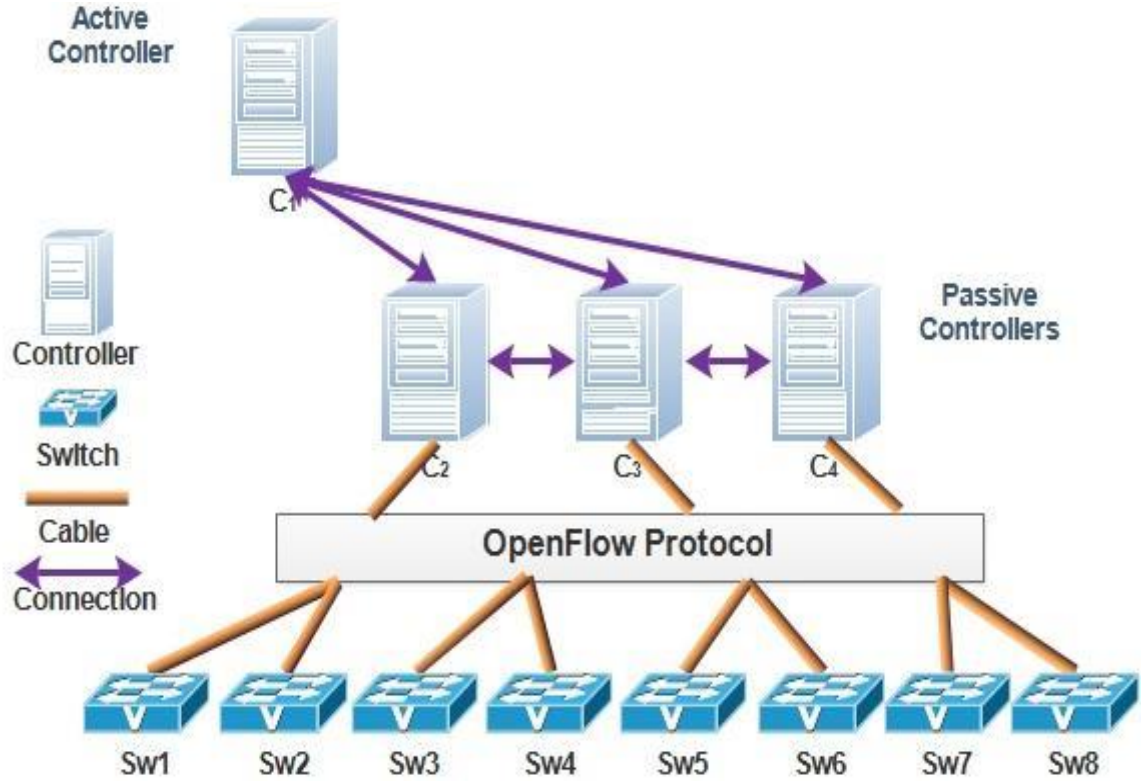


Figure 4.1: Active replication with multiple controllers

As shown in Figure 4.1, C_1, C_2 C_3…. C_n symbolises several multi-controllers that are deployed in this architecture. The communication between the controllers is done in a distributed fashion, whereby information about their network state is shared or synchronised in real-time using check-pointing to eliminate the single point of failure issues caused by a centralised controller. In this scheme, a group of N controllers are actively connected to various switches in the data plane. In this case, the scheme allowed controllers to share network updates among themselves, and whether a decision was needed was shown in the control plane. The controllers use the information or packets sent from the switches to provide a global view of the state of the network and updated information about the state of the network by sharing information regularly in a synchronised fashion. Moreover, each controller has a unique controller ID assigned to it.

Since each controller processes the same events and maintains the full network state to preserve state consistency, the processing capacity of the control plane is often independent of the number of controllers. Therefore, the design principles for the multi-controllers of the proposed FTM are as follows:

- ***Reliability***: The deployment of FT is used to avoid a single point of failure in the event of a network failure. This allows the network to be continuously active and allows the controllers to manage and control the network permanently. In terms of reliability, the number of controllers is important so that in the event of the failure of the primary, another controller is selected immediately to take charge of the network [93].
- ***Availability***: The implementation of the SDN's FT and multi-controller deployment with active replication mechanism are used for fault detection, recovery process and synchronisation to maintain a constant good operational state of the system [94].
- ***Consistency***: Controllers and nodes are placed in a distributed fashion using an active replication mechanism to guarantee continuous control and update of network state in real-time among controllers if there is a fault. That is, all controllers are synchronised at the same time and have the same global view of the network. In this case, when the primary controller fails, the selected controller takes control of the network immediately without the need for a rollback operation [93].

4.3 Proposed fault-tolerance model

This section presents the proposed model known as the FTM, which is made up of two major components: controllers and a stable database (sDB). As stated above, the controllers' design of the proposed FTM is based on multiple controller architecture and an active replication FT scheme. The rationale is to achieve reliability, availability, and consistency among controllers devoid of checkpointing and rollback to make the SDN model free of a single point of failure. The proposed model for fault tolerance is shown in Figure 4.2. To achieve this, the system applies constant fault monitoring and detection approaches and recovery to accomplish a fault-tolerant system. In the proposed FTM design, several SAs denoted by SA_{C1} and SA_{Cn} are used to monitor the status of the controllers by establishing continuous communication between them, to determine whether a controller is dead or alive through message exchanges. The stable database (sDB) is where all global network information and events are stored and updated periodically. The information stored in the sDB assists the system in the fast fault detection process by comparing the information stored on the current events to identify whether the fault is a node or link. The types of faults that are stored in the sDB are controller failure and network link faults. The design structure is presented in Figure 4.2.

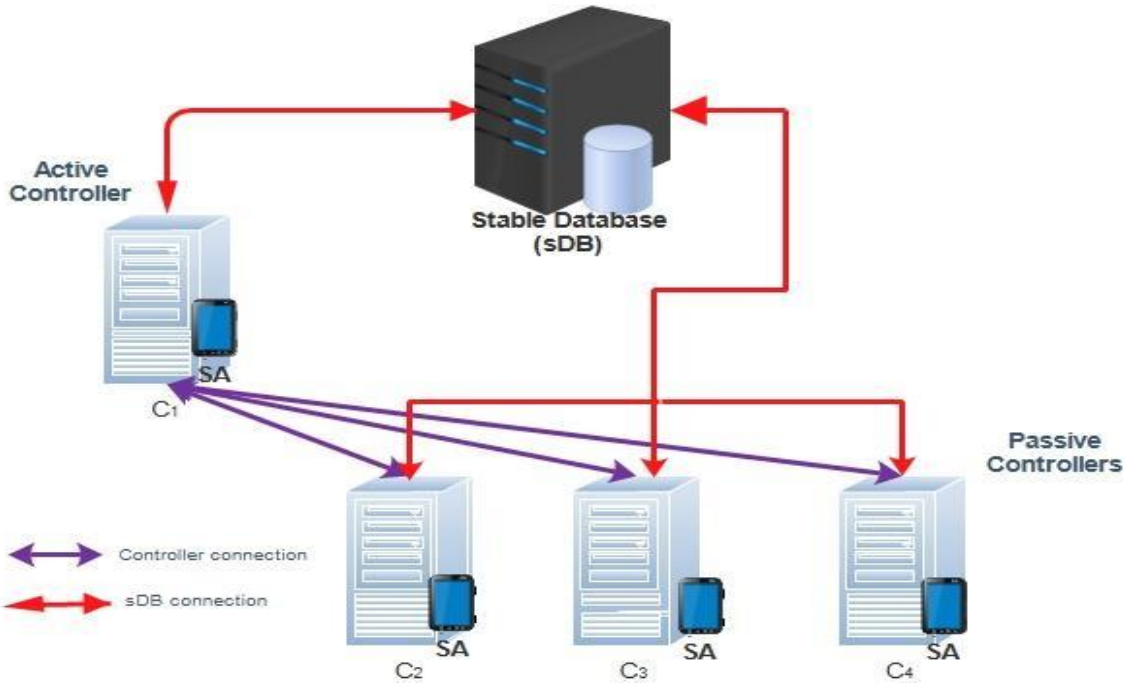


Figure 4.2: FTM design

4.4.1 Controller architecture and placement

The architecture of the controller plane determines whether a single controller or many controllers are used more commonly in the design of the controller plane. One controller architecture is used, and although it has the advantage of centralised network control the challenge continues to be identifying a single point of failure that might bring the entire network down. On the other hand, the multi-controller architecture overcomes such a challenge but introduces a new challenge of deciding on how many controllers the network needs to install and where they must be placed, which poses a big challenge to network engineers. This section then presents the controller architecture in the proposed FTM and their placement strategy.

A. **Controller architecture**

In the proposed FTM, the controllers are distributed with a ring configuration so that each of them provides a continuous global view of the network status. Figure 4.3 displays the controller design scheme for the FTM denoted by $C_1, C_2, C_3, C_4, \ldots\ldots C_n$, which is their ID in a circular configuration, whereby *n* represents the maximum number of controllers that can be installed in a multi-controller configuration of an SDN.

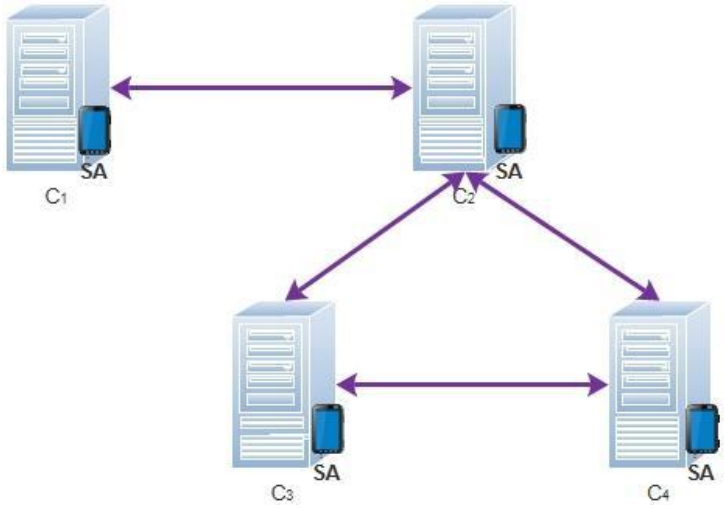


Figure 4.3: SA-enabled controllers and interaction

Moreover, each traditional SDN controller is integrated with a SA_C as shown in Figure 4.3. The role of SA_i is to keep continuous communication between controllers and communicate with other nodes in the SDN environment. The SA constantly monitors the state of the controllers by sharing information among them as well as the controller's network state for potential errors or faults and then that information about those statuses is regularly updated in the sDB. SA has the role of monitoring the state of the controller that houses it while exchanging information about their state by communicating between them with their respective SA_{c1}, SA_{c2}, …… SA_{cn}.

Only one controller (i.e., primary) serves as the lead controller at any one moment and oversees leading and administrating the network. For a newly elected controller C_{im} to take over in the event of a problem, the primary controller C_i and a second controller, C_n, communicate their state with additional backup controllers in a cyclical fashion utilising an active replication process.

B. Placement scheme

One of the objectives of this research is to determine the ideal placement for controllers' deployment in terms of their position and location to improve controller performance during fault tolerance [16]. For the evaluation of the proposed FTM, the deployment of the controller in the medium to large-scale network based on UCPP was selected as the best approach because it considered load balancing and other crucial metrics used by the controller. Because the UCPP

has an unlimited capacity for controller performance in the SDN network environment, it was chosen for the proposed FTM's controller placement design [95]. The advantage of employing UCPP for controllers' placement is that it does not take capacity into account as a performance limitation, and the burden placed on the controllers can frequently be disregarded [95]. The two algorithms k-centre and k-means, which are used in the deployment of the FTM placement were chosen to address the placement issue in this study using the uncapacitated CPP technique [58].

To derive the solution from the discussed placement problem in Chapter 2 section 2.5 and to determine where the proposed FTM would be most effective by measuring the distance between N network nodes, $d_{(i,j)}$, (*see details Chapter 2, section 2.5*) to determine the shortest routes between the deployed controllers [63]. Techniques for the partition methods based on k-means clustering are employed. To lower the maximum end-to-end distance in this situation, the network is partitioned into smaller sub-networks using the k-means clustering-based partition algorithm. To be clear, the initial nodes chosen for clustering algorithms are the "centre", while the actual centre of each cluster is referred to as a "centroid".

There are four main steps in the k-mean clustering algorithm [24]:

1. Initial k-clusters and assign a random sampling-based centre to each cluster.
2. Use Euclidean distance to assign nodes to one of the clusters.
3. Recalculate each cluster's centroid.
4. Continue to follow steps 2 and 3 until each cluster stays the same.

The sum of squared distance, according to the k-mean, represents the overall within-cluster variation. Euclidean distances in the equation between the item and the appropriate centroid are indicated in Equ.4.2.

$$d(C_k) = \sum_{x_i \in c_k} (x_i - \mu_k)^2 \quad \text{.....(4.2) [24]}$$

In this case x_i is a data point that belongs to the cluster c_k, μ_k Is the average point value assigned to the cluster c_k. It shows the shortest route from node $v \in V$ to $s \in V, n = |V|$ which is the number of nodes; S is the number of available controllers and S' number of controllers that should be placed so that $|S'| = k$. The typical delay for controller deployment $L_{avg}(S')$ is modelled as in equation (4.3).

$$L_{avg}(S') = \frac{1}{n} \sum_{v \in V} d(v, s) \quad \text{.....(4.3) [17]}$$

Algorithm 1: *k-means* clustering [24]

Input: node values
Output: cluster formulation

Step 1: clusters centres

Step 2: Allocate the vertex v (v$v(v \in V)$ by applying the relation to one of the K clusters,
whereby $v \in cluster_i$, if $d(v, ci) < d(cj), \forall j \in \{1,2, \dots, k\}$ whereby $d(u, v)$ represents the route connecting the nodes that are shortest $u \; and \; v$.

Step 3: Update centroids $C' = \{c'^1, c'^2, \dots c'_k\}$ such that the total of the shortest paths between all of the cluster's points and the new centroid c'$_I$ is minimised.
$C'_I = v_m, \; if \; d(v_m, v) = minimum$
And $\forall m \in size \; cluster_i, v \in cluster, I = \{1, 2, \cdots, k\}$

Step 4: Use steps 2 and 3 repeatedly until no clusters change

The average latency is optimised using the k-median method. Finding the centre of a k-cluster utilising the k-median is a clustering method that reduces the total distance between the cluster's centre and all other locations [103]. We used k-means and k-medoid clustering to tackle this problem [103]. To discuss where controllers should be placed inside the suggested SDN-FTM architecture, we made use of the *cityblock* distance metric in MATLAB using the algorithm to calculate the total absolute difference *x* plus the distance in *y*. The denotation and calculation for the *cityblock* distance with k dimension between two points, *a* and *b* are as follows:

$$\sum_{j=i}^{k} | aj - bj | \quad \text{……(Equ.4.4) [96]}$$

To compute the distance using different distance metrics and to find out the centroid clusters between the controllers and switches whereby x represents the distance and c the centroid as presented in the below equation 4.5:

$$d(x, c) = \sum_{j=i}^{k} | aj - bj | \quad \text{……..(4.5) [96]}$$

The primary goal of utilising the equation above is to calculate the k-means clustering technique to specify the placement of the controllers and their distance from one another. Following is a presentation of the algorithms:

1. K-median algorithm

This solution enables the researcher to find C_n. The suggested FTM proposes a set of controllers C that reduces the average propagation latency nodes (switches and controllers) using *d (V, C_n)*, from equation 4.1.

$$L_{avg(C')} = \frac{1}{N}\sum_{v \in V} \min d(v, C_n) \quad \text{.....(4.6) [96]}$$

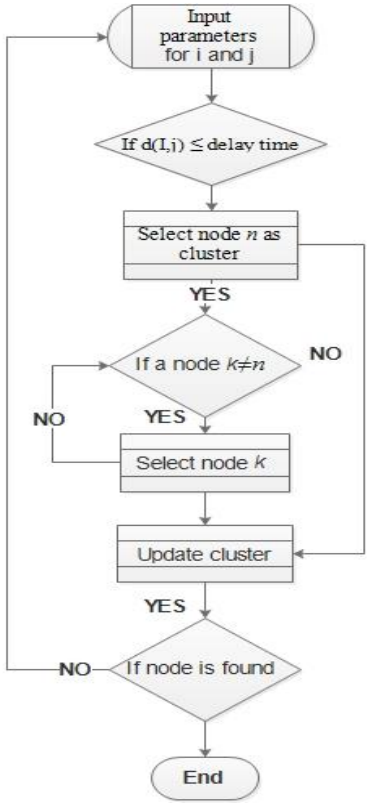


Figure 4.4: K-median algorithm [97]

The shortest link or connection in the network is determined using the k-media solution which is $d_{(i,j)}$ by choosing a node represented by *n= [I, j]*. Through this approach, the average latency *(L_{avg})* of the remaining network infrastructures can be reduced to a minimum. The k-median algorithm is shown in Figure 4.4 [97]. To reduce the network delay, the average latency problem between nodes is solved using the k-median method [97].

2. K- Centre algorithm

To better arrange the controller in the proposed FTM, the greatest separation between the nodes denoted by *v* and the closest controller C_n is minimised using equation (4.7).

$$L_{k\ -\ center} = \max_{v \in V} \min_{Cn \in C} d(v, C_n) \quad \text{.....(4.7) [96].}$$

The *K-centre* solution is employed to identify the ideal place for controllers' placement on the SDN network infrastructures for the proposed FTM. The K-centre algorithm is presented in Figure 4.5.

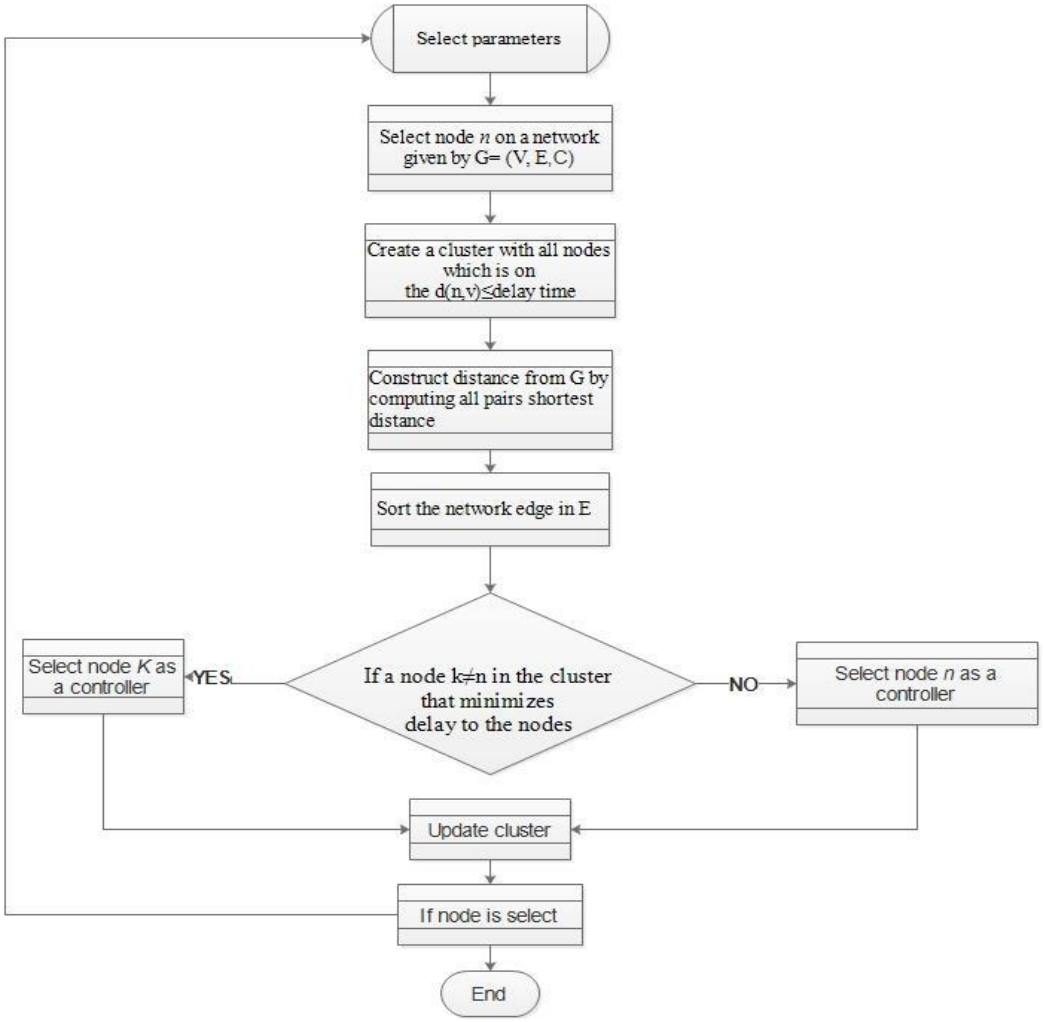


Figure 4.5: K-centre algorithm

1.4.2 Stationary agent functionality

The SA introduced in this research, particularly, in the proposed FTM, is a program developed in Python that runs in both controllers to maintain continuous communication between the

controller for monitoring and detecting possible faults or failures in the system. For effective monitoring and reliability, the proposed SND-FTM, and respective SA that are housed in the controller communicate between them to establish constant communication. SA continuously checks the state of the controller by sending periodic information in 40 seconds to all nodes on the networks and checks the returns response to detect whether the controller is ALIVE or DEAD. The main responsibility of SA is to establish communication between the controllers and take information about the state of the controller that houses it and share it with the sDB which is in charge of storing all the information of each controller's state on the network.

As one endpoint to two-way communication between the controller link applications utilising a network, the SA serves as a socket in the SDN network platform. A technique of network-based inter-process communication is provided by this SA mechanism. The SA also makes use of the sockets system call to establish communication between them, much like pipes are created with the pipe system function. In this way, it offers a network-based bidirectional first-in-first-out communication. Each SA is assigned a unique address of the controller that housed it. This address is made up of the SDN port number, the controller ID and the IP address. Figure 4.6 presents the SA communication between controllers.

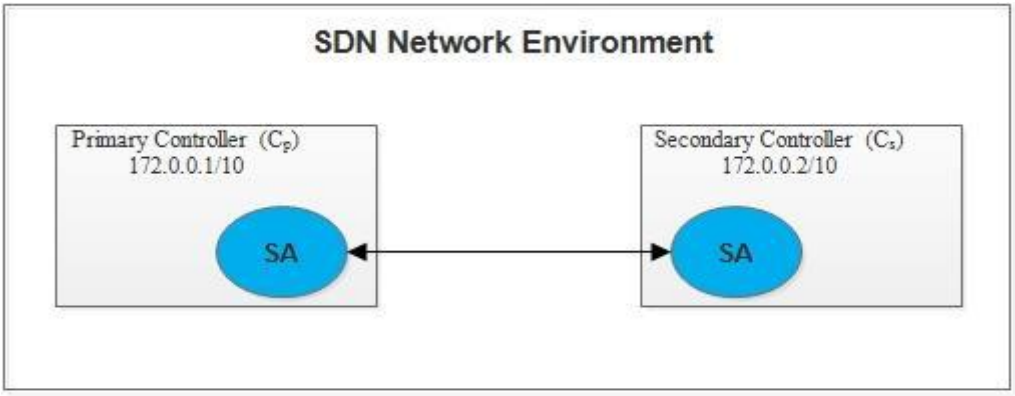


Figure 4.6: Communication between SAs

SA broadcasts the type of network socket with explicitly defined procedures for creating and terminating connections as well as for detecting mistakes or defects. Communication that offers an unrestricted-oriented, sequential data flow without record boundaries is known as connection-oriented communication in the controller deployed in the proposed SDN-FTM network platform. This approach is comparable to communication between two phones and a conversation (data transfer) occurs after the phones (on opposite ends) establish a connection. Within this context, the primary controller (C_p) creates a socket (request message) and sends it to the SA house in the secondary controller (C_s) using SDN Port addresses and then awaits a

response. The SA housed in the secondary controller (C_s) in turn creates a socket (reply message) and then tries to establish a connection with the controller in charge (C_p). Once the connection has been made, the allocation of information about the controller's status takes place. Figure 4.8 shows the structure and method used to demonstrate the function of the SA.

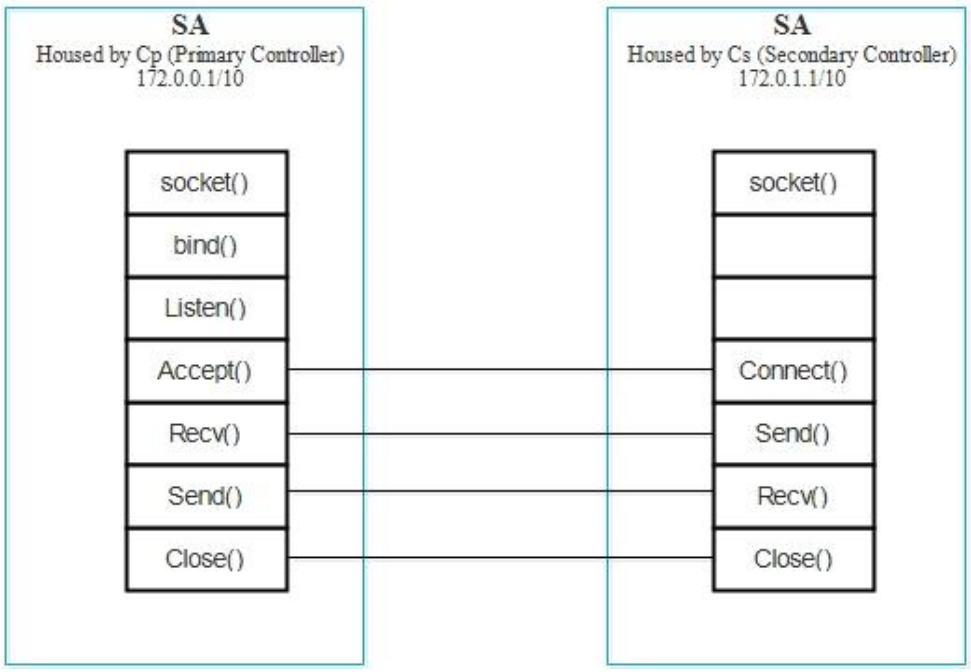


Figure 4.7: SA structure and methods

As shown in Figure 4.7, the SAs are housed in the controllers (primary and secondary) and Their purpose is to act as listeners on a specific SDN port at an IP address, while the other reaches out to the other to establish a connection in some way. The one in the primary controller forms to the listener while the secondary reaches out to respond to the request. Table 4.1 provides the SA function call or method and description as per its design.

Table 4.1: SA function call and description

Function call	**Description**
create()	making a socket
bind()	It is an identification of a socket (like an IP address)
listen()	ready to connect with an entity
connect()	prepared to send messages
write()	To send data
accept()	Confirmation, to accept to receive a message from a sender
close()	To close or end a conversation

The SA used within the context of this research is developed using a Python program and it runs in both controllers to maintain continuous communication with each other. It has three major methods, namely: the sender, the receiver, and the action of the SA.

Sender (): This method allows the agent to create, request and identify the controller that housed the SA and establish the connection between them.

The process by which the SA sends requests is described in the section below.

Algorithm 2: SA Sender Method

```
    Input: signal message
    Output: request-reply
1.  From Django. dispatch import Signal Import requests
2.  my_signal = signal(
3.              providing_reply=['my_reply',' request']
4.                      )
5.              def_CntrID_send(request):
6.              my_signal.send(
7.              sender=none,
8.              my_reply=" Hello",
9.             request=request
10.  )
```

ddd

Algorithm 2 shows the sender method we started by importing the signal and request library in Python and making it allow the creation of the signal to be sent by the SA. After providing the arguments request confirmation of the reception of the request by the received node, we defined the parameter *CntrID*, which represents the controller hosting the SA. Once the signal was sent, it identified the controller that housed the SA and established the connection between them. If the controller was up, a reply request was sent back, and if the controller was down, no response was sent. In this case, there was none because the sender in this instance was not an object and the receiver could do anything with it, hence it was set to none.

Receiver (): allows the SA to listen to incoming requests using controller ID which is the link to the IP and the SDN port. Algorithm 3 presents the SA receiver method.

Algorithm 3: SA Receiver Method

```
    Input: signal message
    Output: request-reply
1.  From Django. dispatch import Signal Import requests
2.  my_signal = signal(
3.          providing_args=['my_arg',' request']
4.                  )
5.          def_CntrID_send(request):
6.          my_signal.send(
7.          sender=request,
8.          my_reply=" Hello",
9.      )
```

Action(): This method allows the SA to operate and work in its environment as well as the controller that housed it. The SA action method is to check the status of controllers in the network, by interacting with other SAs. Each state of the controller is saved and updated to the other host in the network regularly to ensure that the network is running continuously. For the SA to operate and work in its environment as well as in the controller that housed it, we use the method in Python for that functionality and this is shown in Algorithm 4.

Algorithm 4: SA Action Method

```
    Input: signal Messaging
    Output: request-reply
1.  From message sender import MessageSender import Signal
2.  Credentials = {
3.          "Signal":{
4.          "port": 6633
5.          "Controller ID":"CntrlID",
6.          "sender_SA":" SA",
7.  },
8.  }
9.  Method = "signal"
10. message = {
11.         "controller status": "ALIVE",
12.         "status": "The controller is available.",
13. "response": "ALIVE",
14. "senders": {"controller info": "controller, IP address"}
15. }
16. MessageSender (credentials= credentials, method=method).send_message("controller
    Status",message=message)
```

1.4.3 Stable database (sDB)

All global network information and events are kept in the stable database (sDB) and updated regularly. By comparing the information stored in the current to that recorded in the sDB, the system may quickly determine whether a node or connection is at fault. Figure 4.8 presents the SDB design.

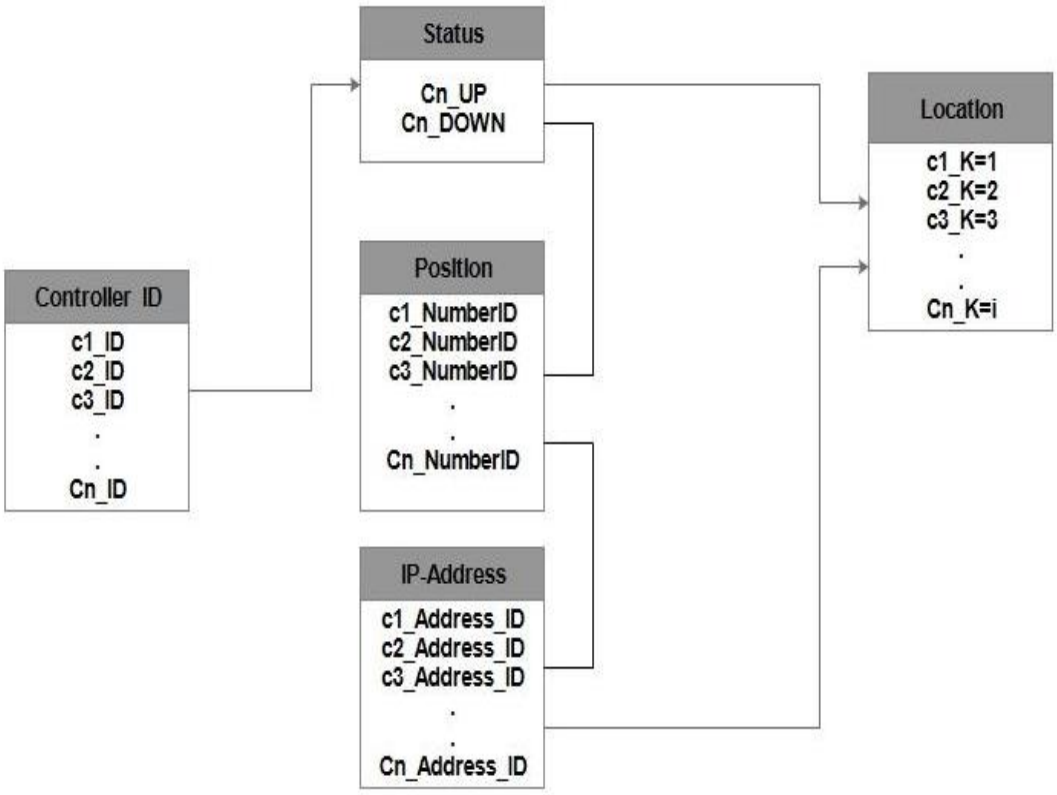
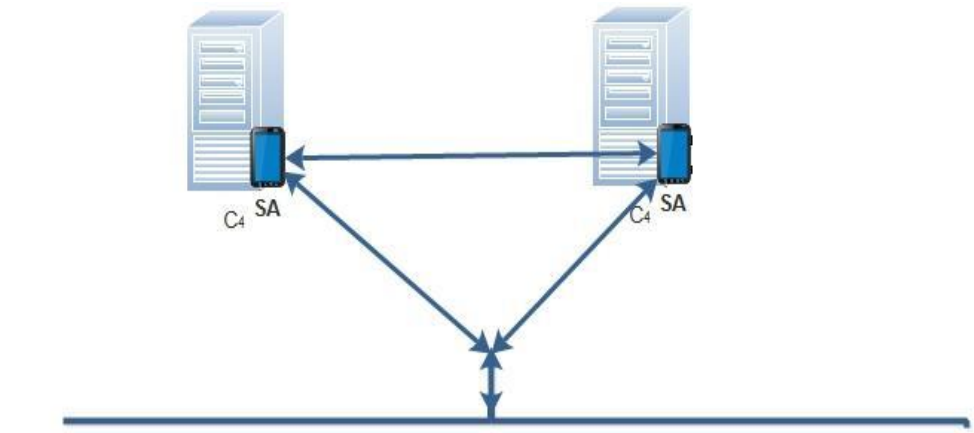


Figure 4.8: Stable database design

The sDB stores information such as controller ID (Cnt_ID) which displays all the details of the controllers, and the controller IP address that specifies the controller position on the network so that each controller can be identified in the case of a fault, controller state (Cont_State) that defines either the controller is UP or DOWN, controller location (Cnt_loctn) that defines the position of a controller on the network facilitated placement, time delay (Time_Dlay) that shows time delay of the controller to respond when connecting with SA as a well as response times that is responsible for recording all the successful response time (Rsp_Time) for the controller to communicate with the SA. Figure 4.9 presents the process involved.



Cnt_ID	Cnt_Position	Cnt_Status	Cnt_Location	Time Delay	Resp Time	Cnt_IPAddress
C_1	1	UP	k=1	0:35	0:10	127.0.0.1/10
C_2	2	DOWN	k=2	1:05	0:20	127.0.1.1/10
C_3	3	UP	k=3	0:10	0:35	-
C_n	n	UP	k=n	0:41	1:00	-

Figure 4.9: Stable database communication with SA

4.5 Fault management

In the design of the proposed FTM, assume that a fault is considered the root cause of an anomaly. When a problem occurs in a system, an error results in a fault in the system and a failure is reflected as the outcome whereby the system is down. The type of faults that the system detects include controller faults and nodes/links faults whereby the entire accessibility is unavailable [3].

i. *Controller fault* happens when the controller does not identify or receive a request for an incoming message from the secondary controller or fails to send messages to respond to the primary controller request or demand in a space of 40 seconds [3].

ii. A *node or link fault* occurs when a controller does not receive a reply message request in 40 seconds from the SDN port or an incorrect message in response to the controller message. The assumption is that the link is down. If the reply comes in the space of 40 seconds, we assume that the link is up [3].

All activities executed during the process of fault management in the FTM guaranteed the network's dependability and accessibility. The proposed FTM applies three stages to achieve fault tolerance, namely controller synchronisation, detection, controller selection and recovery. Figure 4.14 presents all the phases that are involved during the detection and recovery process [98].

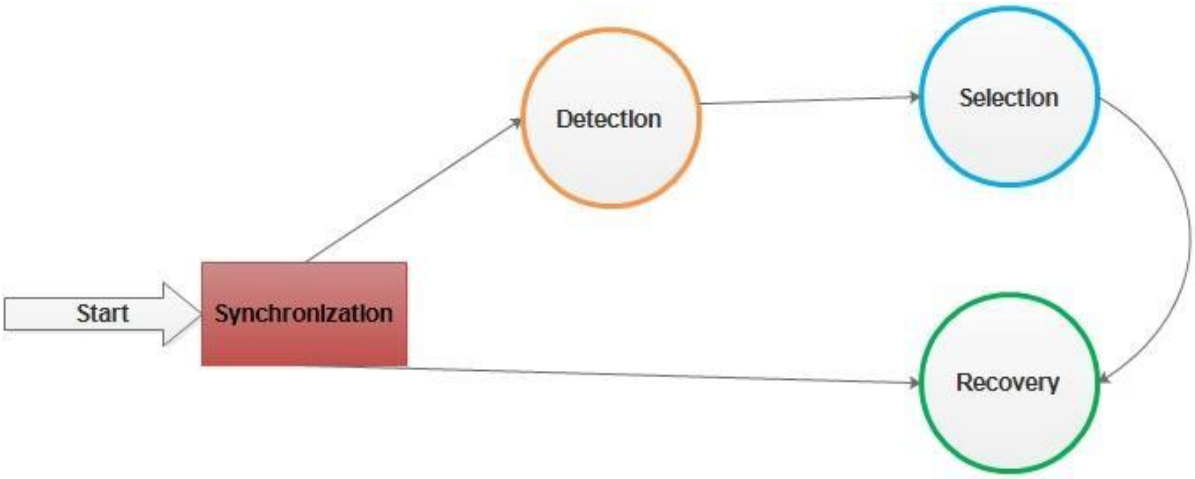


Figure 4.10: Fault management phases

Synchronisation: When the network is initialised, the primary controller requests role assignment information to the secondary controller and the information is shared with all the controllers using a communication process done by SA that is housed in the controllers. Using synchronisation this information was shared with the entire network to notify them about the primary controller and the state of the network.

Selection: Depending on variables like load and latency, the secondary controller in the case of a fault is chosen. During a fault, the SA checks the distance between the controller using information stored in the sDB to select the controller with less load and low latency to take charge of the network as primary. If the controller's load during a failure surpasses its capacity due to additional load from switches and the controller also fails to receive assistance from others during load balancing, the controller was not chosen as the primary controller. The communication distance between the controllers during placement determines the latency between them. For example, if the distance between controller c1 and c2 is greater that shows that there is higher leniency, in that instance, controller c1 was not chosen to serve in the capacity of the primary controller, rather, c2 was chosen as the primary. This process is done using the information stored in the sDB by the SA monitoring activity.

Recovery: To carry out the recovery process of the proposed FTM, the primary controller sends regular request packet messages in 40 seconds out to a specific port on the SDN network, to guarantee the recovery process. Once the reply from the switch is back, SA carries out the task of gathering data from all nodes connected to the SDN network environment and updating that in the sDB. This process is based on proactive approaches. This allows the restoration, recovery, and protection of network services and applications from failures by allocating paths dynamically after the selection of the new controller.

When the primary controller malfunctions, the switch demands new data from the nearest controller using analysed network controller port statistics to identify to closed link that is up to direct the message to it to assign the flow rules of the network. Throughout this procedure, SAs housed in other controllers are also updating their information and communicating that between them, which, in turn, is stored in the sDB. Using the load balancing and latency during placement, the selection of a new controller was done, and its role was assigned to the rest of the controllers in the network. Once the new controller becomes active then the whole process is repeated. In the event whereby the controller is down, the system analyses network statistics to calculate the distance of the nearest controller using latency and controller load balancing to assign a new controller the role of the primary to manage network resources.

The architecture of the FTM-SDN adopts multiple controller designs using distributed approaches whereby all controllers have a connection between them and share all information in the network. In the proposed FTM, only one active controller plays the role of the primary by managing the network connection as it updates the other secondary's current status controller in the network by applying the state replication technique. A collection of data is kept by the network controller using SA and sDB. SA has the role of monitoring the state of the controller that houses it while exchanging information by communicating with their respective $SA_{c1,}$ $SA_{c2,}$ $SA_{c3,}$ SA_{c4} … $SA_{cn.}$ The sDB is where all the information about the network and controllers from there is stored and updated regularly. Figure 4.11 illustrates the design of the SA communication flow.

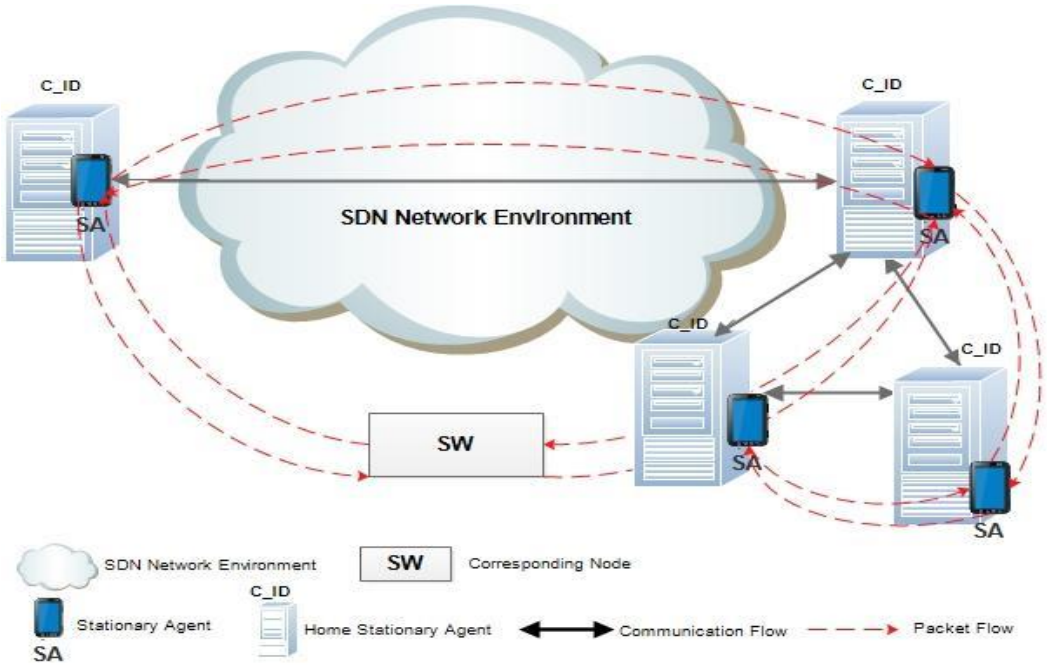


Figure 4.11: SA communication flow design

4.5.1. Monitoring and detection

Using the OpenFlow protocol, network statistics requests handle the monitoring and detection of network faults, which might affect hardware or links. The OpenFlow protocol of the SDN platform provides several message kinds, including controller-to-controller, asynchronous, symmetric, and controller-to-switch, each having numerous sub-types. Using the SA, the controller keeps track of communication between them. The responsibility of the SA is to monitor the process of communication between the controller using the controllers that house it while exchanging information with their respective SAs. In the process of monitoring, the SA uses an exchange message between them at the interval of 40s, to keep track of their status, if the SA in the primary controller sends a hello request, and receives the reply in the interval of 40s, that indicates the controller is ALIVE; and if the backup/secondary controller does not respond in the interval of 40s, then the controller is assumed dead.

In addition, to fault monitoring and detection, network statistics request techniques to identify faults that can occur in the network.

1) For hardware (controller/switch) faults, the primary controller establishes communication using an OpenFlow protocol connection by sending a “HELLO” request message to all the secondary controllers in the network. Upon the reception of that message, the secondary controller replied to that request in 40 seconds. If the reply is not received by the recipient

controller, then the controller is down. If the receiving recipient replies to the request in 40 seconds, then the controller is up.

Assuming that the deployment of a switch and controller may be expressed as an N x M binary matrix Ω [94], a switch may select only one controller as the primary controller to satisfy the dynamic limitations. According to Equation (4.8):

$$x_{i,j}^{t} = \begin{cases} 1 \; i^{th} \; the\ switch\ is\ connected\ to\ j^{th}\ controller \\ 0 \;\; others \end{cases} \quad \text{...(4.8) [94]}$$

Assuming that A is the controller processing capacity, depending on the CPU, bandwidth, and memory. The value of A is represented by A = [Ai……An], whereby M is the number of network controllers, the controller is denoted by C and represented by C = [Ci……Cn]. L_M characterises each controller's redundancy factor which varies from 0 to 1. This indicates that the controller has enough capacity to carry out state synchronisation. A total of all switch requests for flow that are connected to all the controllers j^{th} is the flow requests that it needs to handle in time *t*, which is denoted in equation (4.9) by:

$$\Phi_j^t = \sum_{i=1}^{N} \lambda_i^t x_{i,j}^t \quad \text{……………………………………… (4.9) [94]}$$

The flow request rate on the i^{th} switch in time slot *t* is represented by λ_i^t where *ti* varies for each switch. Each access switch in the OpenFlow network sent out requests for flow setup. As a result, during the service period for flow requests, there were several "packet-in" requests coming from switches to the controller. Through the use of a message and OpenFlow, a controller asks a switch for port statistics, and the switch responds with the statistics.

2) In the event of a link fault, the primary controller sends regular request packet messages in 40 seconds out to a specific port on the SDN network platform, if the reply from the port is not sent in that time interval, it is assumed that the link is down. This process is accomplished by using network statistics requests and replies between the controller and OpenFlow.

3) Statistics requests and replies can also be used to detect controller failure and node/link failure during the request:

- When the controller receives a network statistic reply from OpenFlow within the 40 seconds, then it shows that there is no link fault.
- When OpenFlow does not receive statistics requests from the SDN port in 40 seconds, then the controller is faulty or dead.

Otherwise, using a message and OpenFlow, a controller can ask a switch for port statistics and the switch replies with the statistics. The system analysed network statistics using all the network event messages utilising the OpenFlow Port Statistics technique to identify to closed link that is up to direct the message to it. To achieve this, the event handler is created to receive the *PortStatsReply* (Port Statistics Reply) message to ask the switch for a response.

See Figure 4.12 for an illustration of monitoring and detection using network statistics procedure.

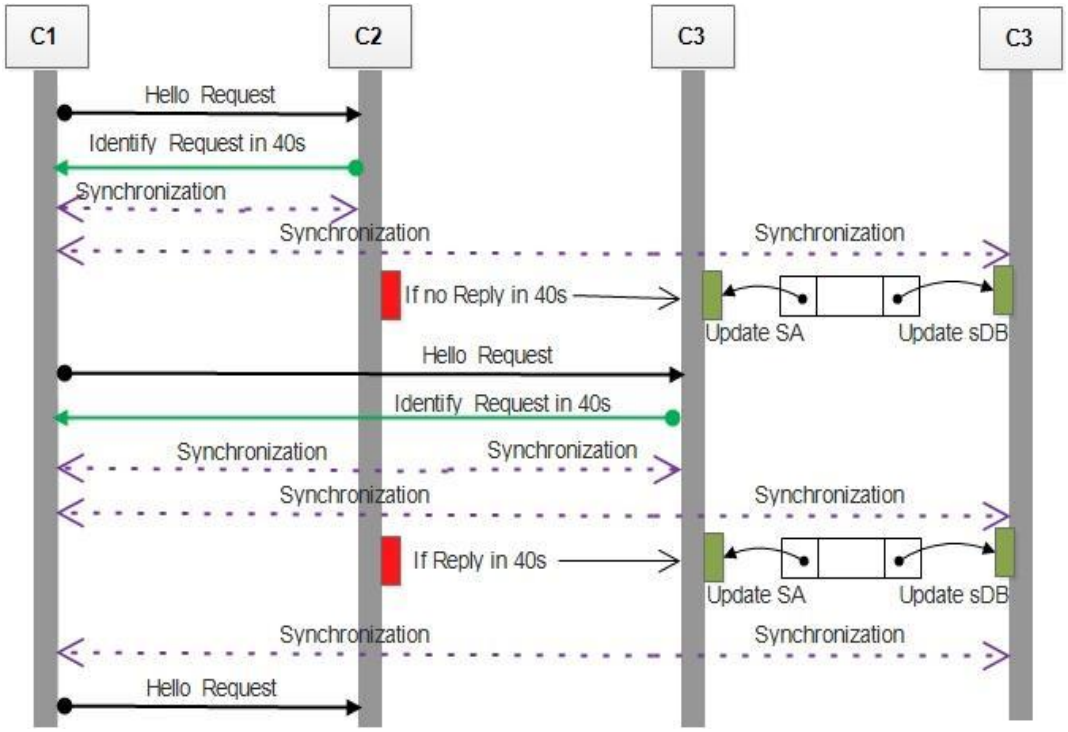


Figure 4.12: Detection using network ports statistics message

4.5.2. Controller synchronisation

As indicated in the previous sections, all controllers in the network have a connection between them and share information. Only one active controller serves as the primary in the proposed SDN-FTM, managing the whole network connection and utilising the synchronisation

technique to update its current state to the other secondary controllers (C_s) in the network. The synchronisation of the controller involves the state replication technique in the controller plane of the designed SDN-FTM. In the course of the network initialisation process, the network information stored in the sDB and SA is broadcast to all available controllers in the SDN platform. In case there is a controller failure or a link failure in the network, its status is communicated by the SA and broadcast to all the other controllers in the network. If the network's link malfunctions, the primary controller (C_p) fails or there is a link fault in the network.

All network controllers can exchange knowledge about the new controller that steps in to serve as the primary following a failure during the synchronisation procedure. Through the OpenFlow protocol and port, the SA and sDB information is communicated as part of the set of data that is exchanged during this procedure. This is presented in algorithm 5.

Algorithm 5: Controller Synchronisation in the SDN-FTM Algorithm

Input: Set of controller C = [C_1, C_2, C_3, C_4, ……C_n]
Initialise: SA, sDB, Broadcast SMR to C
Steps:

```
1        Begin
2          Controller Cp sends "HELLO" request to all controller Cs;
3          Controller Cs reply in the 40s;
4          Synch_SMR;
5          Add info into SA + sDB;
6       If no reply in the 40s {
7            Update new info into SA;
8             Update new info into sDB;
9        Else wait until Cp "HELLO" request is replied to;
10           drop packets from Cp;
11     Do Share Cp info with all Cs;
12         Request for current SMR;
13          Synch_SMR;
14         Update Cp info into SA;}
15      End ()
```

4.5.3. Controller selection

The controllers in the proposed SDN-FTM are connected using a multiple-controller architecture, as was mentioned in the previous section, but only the primary controller (active) is responsible for managing the network. If the primary controller fails, the secondary controller can be selected using load balancing and latency during controller placement. If the primary controller malfunctions, the secondary controller uses the SA to contact the closet controller for the most recent data before assigning a replacement controller per the network's flow rules.

Algorithm 6: Controller Selection in the SDN-FTM

Input: Number of controller Cn, available links **E**
Output: Cn controller selection
Initialise: k Set of controller C = [C_1, C_2, C_3, C_4, ……C_n]
Steps:

```
Begin
  From k Set of controller C = C = [C1, C2, C3,C4,……Cn];
  Compute controller placement;
  Update information into SA + sDB;
 if Cp fails {
   Compute the distance between controller Cn;
    Update info. Into SA + sDB;
While Cn distance is defined;
    Compute Cn Load + Cn delay;
 For each selected controller Cn;
     Do Evaluate info in sDB;
        Compute Controller Cn Load + Cn delay in sDB;
     if controller Cn has Minimum load {
       select controller Cn as the new primary Cp;
            else drop Cn;
              Compute controller placement;
   Update information into SA + sDB;
End ()
```

During this process, SAs located in other controllers are also updating their data and exchanging it with one another; this data is then stored in the sDB. During the placement, controller location and position are defined. The controller's load balance and latency are dependent on the controller's capacity to handle more resources than its actual capacity, and the latency is mostly defined by the communication distance between the controllers in the SDN-FTM. The controller may not be chosen as the primary if the load on it is greater than its capacity due to additional demand and it is unable to receive assistance from another controller in the network; another secondary controller with less load was assigned the responsibility of the primary and took control of the network.

Considering the formula below, we choose whether to switch to migration if the controller is overwhelmed. The average arrival rate (F), propagation delay (D), and the amount of flow table entries (N) make up the switches' load [94] see equation (4.10).

$$w1 * N + w2 * F + w3 * D = CLoad\ (Contrller\ Load)$$ ……(4.10) [94].

Where the weight coefficients w1, w2, and w3 are added together to equation 1.0, they determine the loa of each switch, similarly, based on its entries in the flow table and then calculate the controller's overall load on the number of switches and events. The primary

controller's load balancing was kept in the sDB. In addition, if the latency increases with a great distance between the controllers, which is a feature that is not needed; in that scenario that control was not selected, and instead a different controller was chosen, that has a better placement. Algorithm 3 illustrates controller selection in the proposed SDN-FTM.

Algorithm 7: Fault Recovery for the SDN-FTM Algorithm

Input: Number of controller C_n, available links **E**
Output: Cn controller recovery
Initialise: Set of controller C = C = [C_1, C_2, C_3, C_4, ……C_n];

```
1   Begin
2     Controller Cp sends "HELLO" request to all controller Cs;
3     Controller Cs reply in the 40s;
4     Synch_SMR;
5     Add info into SA + sDB;
6    If  reply in the 40s {
7       Compute the distance between controller Cn;
8       Update new info into SA + sDB;
9   While Cn distance is defined;
10      Compute Cn Load + Cn delay;
11      drop packets from Cp;
12   If controller Cn has Minimum load {
13      Select controller Cn as the new primary Cp;
14      Request for current SMR;
15   Else   Synch_SMR;
16      Update Cp info into SA;}}
17    End ()
```

4.5.4. Fault recovery

The backup controller takes over if the primary controller fails to make use of network statistics requests to store in the sDB to identify the distance between the controller so that the nearest controller can be chosen for the appointment of the new primary using load balancing and latency. The SA and sDB update their knowledge of the other controller's status on the network during this time. The secondary controller requests role assignments to the closest accessible controller with the least amount of latency and load when the network is first initialised. The closest controller that satisfies the requirement is designated as the primary controller, taking command of the entire network, and others act as secondary. In this process, the state of the network information was shared with all available controllers through synchronisation provided by the network replication and the SA shares that information with the remaining controllers once the network has recovered and updated in the sDB. See above algorithm 7 which illustrates fault recovery for the SDN-FTM. The process involved in the operation of the SDN-FTM presented in this chapter comprises synchronisation, which assists in fault

detection/monitoring, selection, and recovery. The following section explains the entire process in detail.

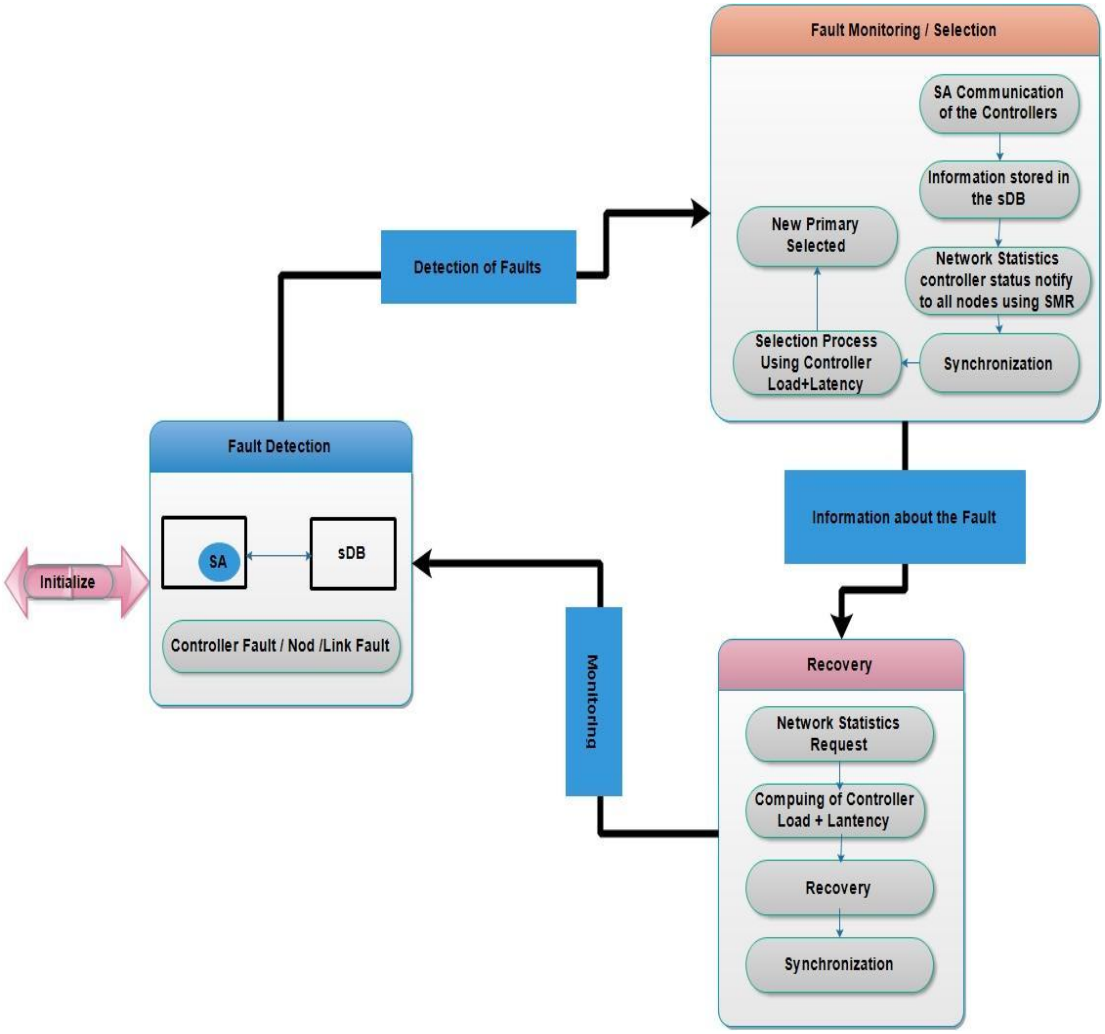


Figure 4.13: Fault tolerance operation

4.6 SDN-FT architecture

Since SDN incorporates the control plane in software as opposed to network hardware, programming is made possible. As a result, the network becomes automated and easier to administer. In the SDN-FT architecture, the controller plays a vital role as it is the central point where decisions such as network configuration, new policies and most of the changes implemented on the network are made. The proposed SDN-FTM applies to small to medium-scale types of networks. However, it can also be modified to be used on a large scale. Its design introduced an SA by modifying the original SDN controller. In the control plane, the SA ensures effective monitoring communication between the controller and all the controllers to accomplish a robust fault tolerance. The purpose of integrating the SA in both controllers is to

ensure continuous communication of the controllers' status of the controller to determine whether it is up or down using continuous exchange messages between them. To ensure that all controllers maintain active connection, the proposed SDN-FTM makes use of an active replication strategy in the control plane to make sure that information among the controllers is synchronised in real-time. In the event of the failure of one controller, the entire network should not be disrupted or stopped, but the rest of the controllers should keep running.

The function of the proposed SDN-FTM is to guarantee full monitoring of the network as well as the controllers in real-time and to detect any faulty components in the system in the event of a fault or failure. Once a fault is detected, the system implements a fault-tolerance mechanism and recovers from the failure within seconds by applying multiple controllers' strategies using active replication to ensure that failure does not affect the operations of the entire network. In the event of a fault (link) or a failure of the master controller, the system elects a new controller to take charge of the primary controller's responsibilities. The selection of the new controller is done by using load balancing and latency during the controller during the computation of network statistics. If the primary controller malfunctions, the secondary controller uses the SA to contact the closet controller for the most recent data before assigning a replacement controller following the network's flow rules. Figure 4.14 shows the proposed FTM architecture.

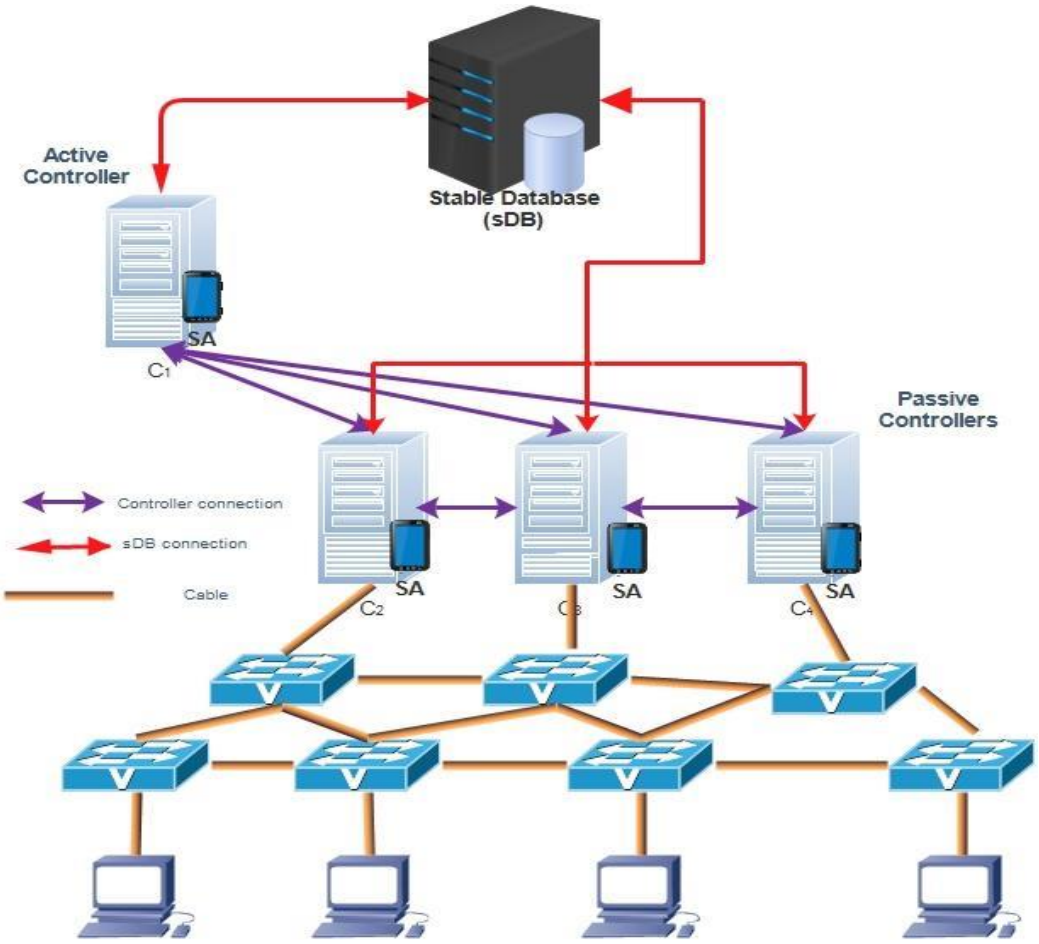


Figure 4.14: Proposed SDN-based FTM

4.7 Design evaluation

The evaluation of the proposed SDN-FTM utilising various scenarios is based on different principles. The following section presents the design principles that were discussed in this chapter's pressing section.

4.7.1 Theoretical evaluation

To analyse and validate the proposed SDN-FTM architecture, there is a need to consider numerous scenarios based on the design principles of an accurate use case, corresponding to a fault-tolerant system. Table 4.2., illustrates various scenarios for the evaluation of the proposed FTM.

Table 4.2: Evaluation of the Proposed FTM

Principles	Description
Placement	● Placement is accomplished by specifying the quality and placement of controllers on the network. ● The SDN-FTM ensures load balancing and minimises latency between controllers. ● To improve network performance and save cost. ● Placement is achieved by the minimum K-centre and minimum K-media algorithms.
Reliability	● The FT deployed a multiple-controller approach to avoid a single point of failure in the event of a fault. ● The system detects faults and recovers them to allow the network to be continuous. ● The use of network statistics requests allows the controllers to control the network permanently. ● The number of controllers is increased and in the event of the failure of the primary, another controller is selected immediately to take charge of the network.
Availability	● The FT introduced multi-controller deployment for fast fault detection and recovery. ● The use of active replication scheme is used for synchronisation to keep the system working continuously in good condition.
Network consistency	● Controllers are connected in a distributed fashion to guarantee constant control and update of network states. ● All controllers are synchronised and share information about each other state in real time using SA to allow the system's functionality in the event of a failure. ● The controllers have a global view of the network to allow rapid selection without the need for rollback operation.

4.7.2 Design comparison with an existing similar system

Fault management in SDN networks was previously covered. They have not given the distributed controller their full attention. We focus on fault management in the distributed controller's control plane. In recent years, several researchers have started placing the focus on the distributed controller, hence this study focuses on fault management in the distributed controller control plane. The following section describes the summarised comparison of the FTM with other similar systems in the literature in Table 4.3.

The proposed FTM was contrasted with various current fault-tolerant methods. Table 4.3 gives an overview of the various frameworks, models and applications from different researchers. Based on the controller and architectural design, the proposed FTM was compared with others. The concept of distributed and multiple controller architectures issued was implemented by BFT-Smart [99], and fault-tolerant and consistent SDN controller, these frameworks demonstrated the availability of services. The master-slave architecture that the Rama framework in [99] is based on is not ideal in a completely distributed network platform. The use of a common database sharing by the master-slave controllers in the fault-tolerant and consistent SDN framework [99], may have reliability and availability problems as a result of the shared database failing. DSCA [106], and FFDC [100] frameworks show that the master-

slave architecture framework has also been used for fault tolerance in SDN. It was shown that in distributed SDN, the failure of the slave controller might impair the fault-tolerance system. Other strategies, like BOND [101], FTLink [102], and smart Routing [102], attempted to fix problems with the network's data plane links. The model presented in [102] allows for simple program reasoning even in the face of errors, and it is implemented via a compiler that is designed to cope with in-network quick failover solution offered in current releases of the OpenFlow standard. The design provides techniques for converting FatTire programs to OpenFlow switch configurations [103]. Contrary to [61] who introduced a platform for a fault-tolerant SDN controller that executes the control messages exactly once per transaction for both switches and controllers. The primary finding of the Revana is that without involving the switches in a complicated consensus mechanism, the state of replicated machines can be improved with straightforward switch-side methods to ensure correctness.

Moreover, the Smart Light in [36] used extra hardware – more controller replicas to enable the system to tolerate controller loss due to faults. Primary backup replication, also known as passive replication, and state mechanism replication, known as active replication, are the two main replication mechanism strategies used for the system. One primary replica in the primary-backup replication model conducts all client0-issued activities and regularly drives state updates to the backup replicas. These duplicates continue to keep an eye on the primary controller so that if it fails, one of them takes over and runs the network. However, none of those techniques accounts for the breakdown of the controller plane. In contrast, the proposed FT-SDN employs several distributed controllers that are modified using a Stationary Agent (SA) and operate independently of one another. The information in the sDB can be stored without any changing of any protocol or using a shared database. The fault management phases of the proposed FT-SDN model are explained in Figure 4.12 and it is the result of its distributed architecture with multiple modified controllers with SA, which was evaluated using the design principles presented in Figure 4.2.

Table 4.3: Comparison of the proposed FTM with existing similar system

Reference	**Type of Framework**	**Focus layer**	**Type of controller**		**Type of architecture**		**Type of application**
			Single	Multiple	Centralised	Distributed	
[36]	CORONET	Data plane	X		X		Enables applications to communicate with OpenFlow switches through APIs and relies on the LLDP protocol to
[103]	FatTire	Data plane	X		X		Programming at the application layer for fault tolerance
[96]	AFRO	Control plane	X		X		Recovery technique for controller failure
[104]	BFT-Smart	Control plane		X		X	Protocol for client requests for
[36]	SMaRtLight	Control plane		X	X		Multiple controllers can share a single data store.
[104]	Ravana	Control plane		X	X		Protocol for managing control messages
[96]	Rama	Control plane		X		X	Choosing a controller based on the Master-Slave architecture
[104]	Fault-Tolerant and Consistent SDN Controller	Control plane		X		X	Database sharing between Master-Slave Controllers
[96]	DSCA	Control plane		X		X	Architecture with Master-Slave
[9]	FFDC	Control plane	X		X		Monitoring traffic to spot problems
[105]	BOND	Control plane	X		X		Setting up every routing route in the switch
[104]	FTLink	Data plane	X		X		Detection of link failure
[96]	Smart Routing	Control plane	X		X		Protocol for link failure detection
[36]	SDN-SDWSN	Control plane		X		X	Monitoring and detecting faults using heartbeat messages and recovering from faults using checkpointing.
This Thesis	SDN-FTM	Control plane		X		X	Monitoring and detecting faults using SA and recovering from faults using network statistics

4.8 Summary

This chapter presents and deliberates the design of the proposed FTM. It starts with a brief introduction of the system, indicating how multiple-controller architecture has been used in this study to overcome the challenges presented by the single controller. The chapter also discusses assumptions that were taken into consideration to ensure the choice and validation of the proposed design system is fault tolerant as well as meeting all the requirements. The architecture of the proposed design systems is presented as a multi-controller architecture that overcomes challenges posed by the single controller to improve network performance. Various algorithms were developed to taste and evaluate each system's components to ensure the overall operation of the designed SDN-FTM. To conclude, to ensure the system's full capability, performance, and application of the proposed SDN-FTM a comparison with previously existing similar systems in the literature for fault management in SDN networks systems was presented.

Chapter 5

FTM Implementation and Evaluation in Small-Scale Network

5.1 Introduction

Hardware and software tinkering are both involved in computer network architecture. Most current networking methods rely heavily on hardware. Its configuration and installation are difficult. A fault or system error/failure can occur when a system is unable to complete a certain task. It is essential to identify the cause of the failure to discover a fault. System fault tolerance is a crucial component of a resilient network. To guarantee high availability and high reliability in the system, fault-tolerance methods are necessary. However, when it comes to network design, the implementation of fault-tolerant varies depending on the size of the network. Its implementation in a small-sized network might not be the same compared to the medium to large network size. In this study, the implementation of the proposed SDN-FTM in a small network was evaluated using SA, considering the specific characteristics, requirements, and constraints of small networks which are simpler architectures and limited resources. In the context of this study, the small network impact of a single fault might be less significant compared to large networks. The perceived benefits of implementing SDN-FTM for fault recovery in small networks do not outweigh the cost and complexity involved.

Therefore, the vital thing to do is to evaluate the performance of the SDN-FTM using various metrics and design parameters in a small network. The proposed SDN- FTM, is measured using Mininet simulation tools for the demonstration of fault tolerance. The study makes use of a modified version of the controller for both Ryu and OpenFlow packaged in Mininet using phyton code. We have designed network topologies and deployed controller (Ryu and OpenFlow) SDN using Mininet to simulate and evaluate fault tolerance for the proposed system in a small network, and the results from the experiments are presented and discussed in the following sections.

5.1.1 Outline

This chapter presents the simulation of the SDN fault-tolerance model (FTM) and it is divided into two sections. The chapter presents the simulation of the proposed SND-FTM using the architecture developed in Chapter 4 which is consisted of four controllers that are connected in a distributed fashion using an SA. The chapter begins with a brief system overview, gives the

experimental procedures and setup of the simulation used in both scenarios and also explains the type of topology used in Mininet to carry out the simulation. In addition, the chapter also presents various demonstrations of the FTM using Mininet and the presentation of the results.

5.3 Implementation

This section presents the implementation of the proposed SDN-FTM based on the system properties using various customised parameters that suit the design principles and the network topology in Mininet.

5.3.1 Simulation set-up

The characteristics of the system utilised to run the simulations are presented in this subsection. In Table 5.1, the properties of the system are presented. The purpose of the simulation was to evaluate SDN fault tolerance using the designed topology in Mininet and to customise parameters that suit the design principles. The following sections are various properties of the systems that were used during the simulation (Table 5.3), evaluation metrics (Table 5.2) as well and other major steps that were taken during the simulation. Moreover, Table 5.4 presents details of the nodes utilised in the simulations.

Table 5.1: The properties of the systems

System Parameters	**value**
Software	Linux-Ubuntu 20.04 LTS, 64-bit operating system
	SDN Mininet, Wireshark
Hardware	Hard Disk 1 T
	RAM 8G
	Processor: Intel® Core™ i7-45705 CPU @ 4.90G
	Link, Switch, Host
	OpenFlow Controller, Ryu Controller

Table 5.2: Evaluation metrics

Metric	**Description**
Throughput	The total size of information sent each time
Latency	The total delay in the amount of data transmitted

Throughput and latency serve as benchmarks for any network performance. In an SDN platform, the performance of the controller has a significant impact on throughput and latency

within the context of our architecture. The throughput of the controller determines the rate of flow response, and similarly, the controller's latency performance criterion is determined by the time it takes to respond to a flow request. Additionally, since it has an impact on throughput and latency performance, the location of the controller (placement) inside the network is crucial to the network's function. As a result, latency and throughput were appropriate metrics to assess the proposed system.

Table 5.3: Simulation parameters

Parameter	**Value**	**Details**
Controller	127.0.0.1…….127.0.0.4	OpenFlow and Ryu
Nodes	s1-eth1, s2-eth2 (Switches) Name C0 ……. Cn (Controllers) h1……..hn (Hosts)	Nodes are devices connected to the network such as hosts, controllers, switches
Protocol	TCP SSL	protocol used during the experimentation
IP Base	10.0.0.1……10.0.0.2	It identified all the nodes connected to the network
OpenFlow Ports	6633, 6634, 6635, 6636	The port functions as an interface between the swathes and controllers
Link	The link is used to establish a connection between the host, switches, and controllers	different link colours in the defined connection between different controllers

Table 5.4: Node details

Node Types	**Description**	**Details**
Controller	A piece of hardware or software that controls how data and applications move through a network and serves as the network's central nervous system.	● Controller type: OpenFlow and Ryu ● Protocol ● IP range 127.0.0.1/4
Switch	A device that helps the connection of the controllers and hosts.	● legacy Switch denoted by sw1….sw_n
Host	Workstation connected to the legacy switches	● hostname (h1, h2, …hn) ● IP addresses range 10.0.0.1/2

5.3.2 SDN topology

This section presents and explains the type of topology used for the FTM in Mininet simulation. Mininet uses different types of topologies and one can always modify or customise the

topology to suit the needs of the research. By default, the Mininet topology contains one OpenFlow controller, one controller, and two hosts. Looking at the FTM architecture in Chapter 4, there was a need to customise the topology to fit the architecture FTM architecture. From the main design, the customised topology used in this section consists of controllers four, seven switches, and four hosts. This topology is designed to evaluate the proposed FTM model with an integrated SA in a small-scale network; see Figure 5.1.

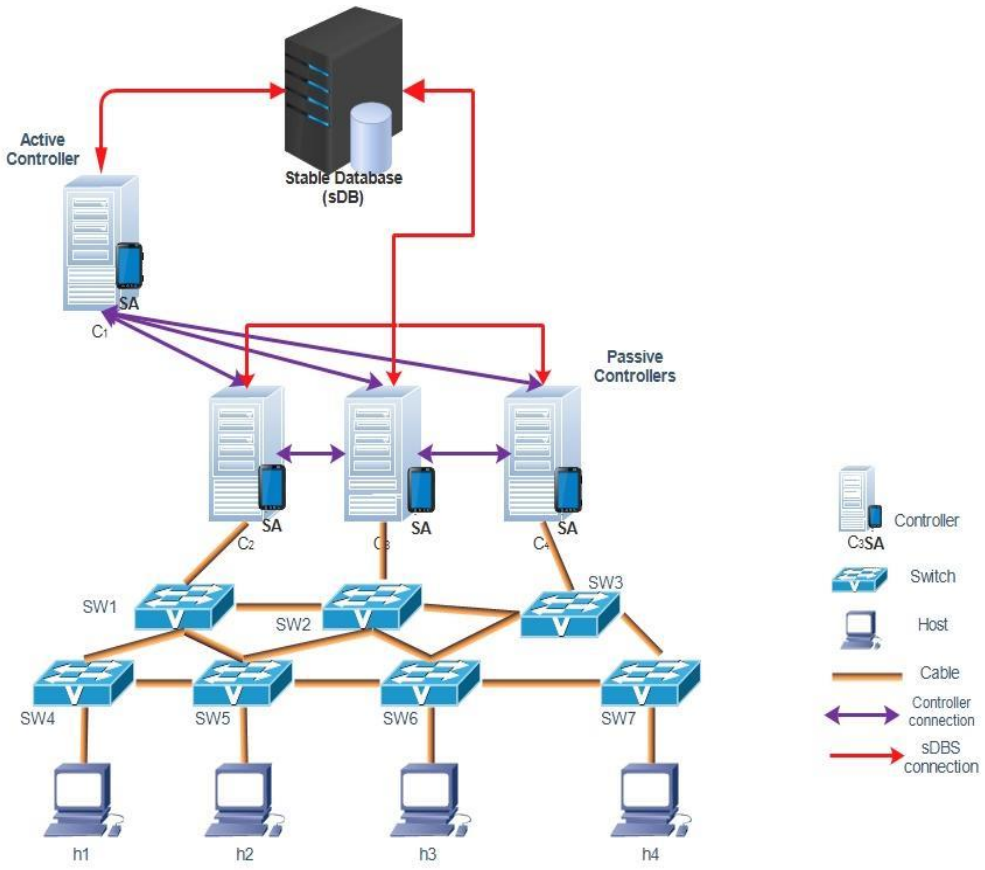


Figure 5.1: Customer SDN-FTM topology

Figure 5.1 represents a custom design for the topology of the FTM in Mininet containing four controllers (Cn), seven switches (SWn) as well as the host (hn)) that is used for simulation in section A using Mininet. The above topology was used only in section A as a standard for small-scale network experiments. Figure 5.2 shows the source code for the customised FTM topology design in the Python Mininet.

Piece of Code 1: Customised SDN-FT topology in Python Mininet

```
{
   “application”: {
      "dpctl": "",
      "ipBase": "10.0.0.0/8",
      "NetFlow": {
         "nflowAddId": "0",
         "nflowTarget": "",
         "nflowTimeout": "600"
      },
      "OpenFlow versions": {
         "ovsOf10": "1",
         "ovsOf11": "0",
         "ovsOf12": "0",
         "ovsOf13": "0"
      },
      "sflow": {
         "sflowHeader": "128",
         "sflowPolling": "30",
         "sflowSampling": "400",
         "sflowTarget": ""
      },
      "startCLI": "0",
      "switchType": "ovs",
      "terminalType": "xterm"
   }
```

When displaying the topology from the command line using the NET test between the nodes connected to the network, the SND-FTF topology representation was displayed in Figure 5.2.

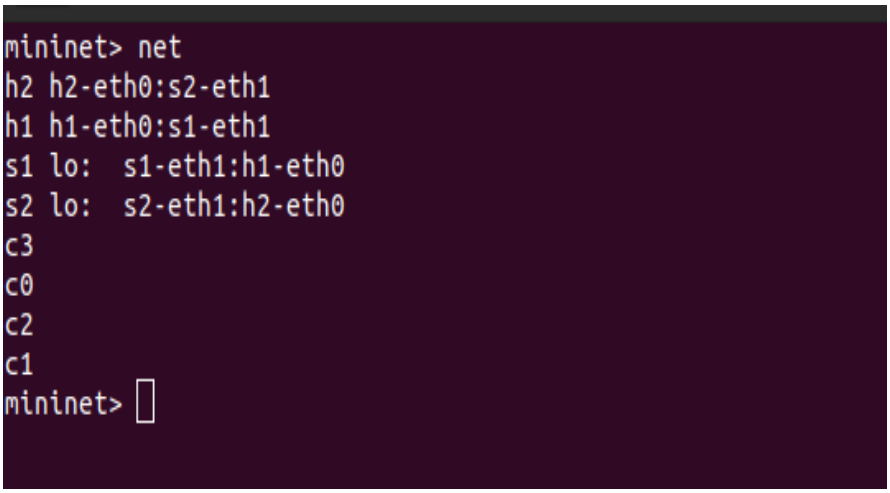


Figure 5.2: Topology design using net commands in Mininet.

5.3.3 Procedure

To simulate the proposed SDN-FTM in Mininet, we made use of two different controllers namely OpenFlow and Ryu with various parameters, metrics, and nodes deployed on the network. As discussed above, the standard-based topology for our SDN-FTM experiments in

Mininet consists of controllers (OpenFlow and Ryu), legacy switches, and hosts. All these nodes used the link to establish a connection between them on the network as well as specific decryption that identified them. To configure the controller, there is a need to define its type, the type of protocol, create its name, IP address and the port number used by that specific controller. Below is the configuration of OpenFlow SDN-FTF controllers in Mininet.

Piece of Code 2: OpenFlow SDN-FT controllers setup in Mininet

```
controllers:
      {
         "opts": {
            "controllerProtocol": "TCP",
            "controllerType": "ref",
            "hostname": "c1",
            "remoteIP": "127.0.0.1",
            "remotePort": 6633,
             "ControllerStatus":"UP",
         },
         "x": "355.0",
         "y": "173.0",
      }
```

The above piece of code represents the basic setup of the OpenFlow controllers used in the experiments. In the virtual machine, we have defined the protocol for the controller, the type of controller used, the name of the controller, the IP addresses assigned to the controller the status of the controller as well as the port that the controller is using. The "x" and "y" values represent the placement between the controllers in the network. This same procedure is repeated several times for each SDN controller setup in the experiment.

As indicated in the previous section, the topology used for this experiment adopts a multi-controller scheme with ten (10) SDN controllers, two switches and two hosts attached to them. To set up a host, we specify the hostname (h1, h2, …h_n) as well as assign IP addresses between the range of 10.0.0.1/2 since we only used two hosts. Host h1 was assigned the IP address 10.0.0.1 and host h2 10.0.0.2. All these hosts were connected to two legacy switches denoted by s1-eth1 and s2-eth2, respectively, as well as all useful links to establish a connection between them. To define the placement between the nodes on the network we used the "x" and "y" values during the experiment. Below is the piece of code illustrating the nodes setup.

Piece of code 3: Nodes' Setup in Mininet

```
"hosts": [
      {
         "number": "1",
         "opts": {
            "hostname": "h1",
            "IP": "10.0.0.1",
            "nodeNum": 1,
            "sched": "host"
         },
         "x": "687.0",
         "y": "481.0"
      },
      {
         "number": "1",
         "opts": {
            "hostname": "h2",
            "IP": "10.0.0.2",
            "nodeNum": 2,
            "sched": "host"
         },
         "x": "227.0",
         "y": "485.0"
      }
   ],
   "links": [],
   "switches": [
      {
         "number": "1",
         "opts": {
            "controllers": [],
            "hostname": "s1",
            "nodeNum": 1,
            "switchType": "legacySwitch"
             "switchStatus":" UP",

         },
         "x": "347.0",
         "y": "331.0"
      }
```

As discussed in Chapter 4 section 4.6, the architecture adopted a multiple-controller design approach whereby all are connected in a circle scheme. Among the connected controllers on the network, only one plays the role of the primary controller and the others are acting as secondary controllers. For this study, we make use of four controllers in our design to conduct our experiment. However, the architecture of the SDN-FTM can support small and medium networks. This approach was based on the rationale to avoid network failure using a single controller and to achieve efficient network management. To address this challenge, this research adopted a multiple-controller approach to overcome this limitation. Our SDN-FTM design consists of four sets of controllers (C1…Cn) interconnected between them so that

secondary controllers can have the network view from the primary, which maintains the network. We have used different colours for the link to show the connection between the controllers themselves and the switches. These controllers used IP addresses ranging from 172.0.0.1/4 for the first scenarios and 172.0.0.1 /10-second ones. In addition, we make use of two switches (s1 and s2) that are connected to all the controllers and share the IP address of the controller that plays the role of the primary, while the IP image of all the secondary is also shared with them. The primary controller (c1) shared network updates with all the secondary controllers connected to it to alert them of any changes on the network using replication as discussed in Chapter 4. In our case, we identify the primary controller as c1 and any chosen secondary controller as Cn. For the simulation in Mininet, we make use of hosts denoted as H_n whereby *n* represents the host number in the application plane layer as devices connected to the switches. The above illustration is shown in Figure 5.1. The following section presents the results and analysis of both scenarios dividing them into two sections, A and B.

5.6 Results and analysis

Referring to the table in section 5.3.1 of the simulation Setup, the connection of these various controllers is made using network connections utilising the IP address and OpenFlow port number for each node. A full summary of simulation notation and parameters is shown in Table 5.3 for both OpenFlow and Ryu. As discussed previously this study used Mininet simulation software, whereby various experiments were conducted, In the first one we created four controllers with SA (OpenFlow and Ryu), with virtual switches, each of them attached to hosts in a small-scale setup network. To evaluate the network performance for all the experiments we used a run-time of 1000 *ms* during the simulation for the SDN-FTM.

To evaluate the SDN-FTM performance, we specified how many controllers were deployed over the networks as well as their number. Defining their positions, guaranteeing load balancing, and minimisation the latency between controllers and switches, to improve network performance and save costs. To ensure availability and reliability, we deployed a multiple-controller approach using the distributed fashion for fast fault detection and recovery using an active replication scheme for synchronisation to keep the network working continuously in good condition in the event of fault or failure. In this way, the controllers in the SDN-FTF guarantee the constant control and update of network states to ensure consensus and network consistency.

5.6.1 Evaluation process

The following section presents the evaluation of the proposed SDN-FTM using a small-scale network platform. This evaluation is based on various scenarios following the design principles of the proposed system.

To demonstrate SDN-FTM, the modified controller discussed in Chapter 4 was used and evaluated in all scenarios. Using the custom design for topology in Mininet, as shown in Figure 5.1, as a standard for all simulations and the communication between nodes in the network was established. The evaluation of the SDN-FTM was based on the performance of the four controllers during the simulations, concerning throughput and latency. We start the simulation by using a small-scale network with OpenFlow and Ryu's controllers running the application on all the controllers using the Python command in Mininet. The command used ensures the specification of only one controller in the network that is chosen to play the role of the primary in the topology. In the simulation, the four controllers are connected to virtual switches which in turn, are linked to a host. Using the standard topology throughout the experiments and changing the controllers' type during the evaluation of fault tolerance, the controllers are running on the virtual machine using IP range 127.0.0.1/4, whereby the first address is used to identify the primary controller in the network. Figure 5.3 uses various colour codes to differentiate the connection between nodes in the network.

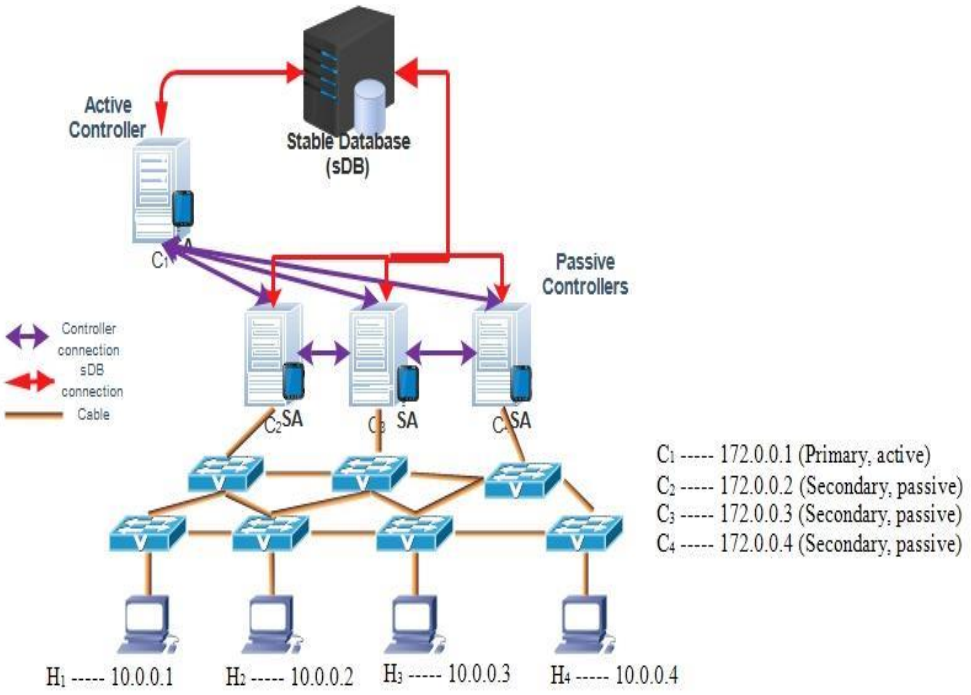


Figure 5.3: Small-scale network evaluation

In this first simulation as shown in Figure 5.3, controller C_1 is selected as the primary (active controller) and is identified and connected to all the other controllers and switches in a circular scheme in the network. This circular scheme connection enables synchronisation so that the nodes in the network can share information about each other state in real time to allow the system's functionality in the event of a failure. In this case, the scheme allowed controllers to share network updates among themselves and whether a decision needed to be taken in the control plane. The controllers use the information or packets sent from the switches to provide a global view of the state of the network, and updates.

In the initial stage of the simulations, the switches' flow table is empty; when using the Mininet *ping* test command from the host machine, the virtual switches sent a packet-in message to the controller to check the data path; The application running on the primary controller responded with the amount of packet transmitted, received, and lost as well as the time it takes. As soon as the switches' flow table is occupied by the correct flow information from all the nodes on the network, then it can establish communication between them. Figure 5.4 shows the illustration of the *ping* test command in Mininet from the host (h1) to the primary controller (C_1 with IP address 127.0.0.1).

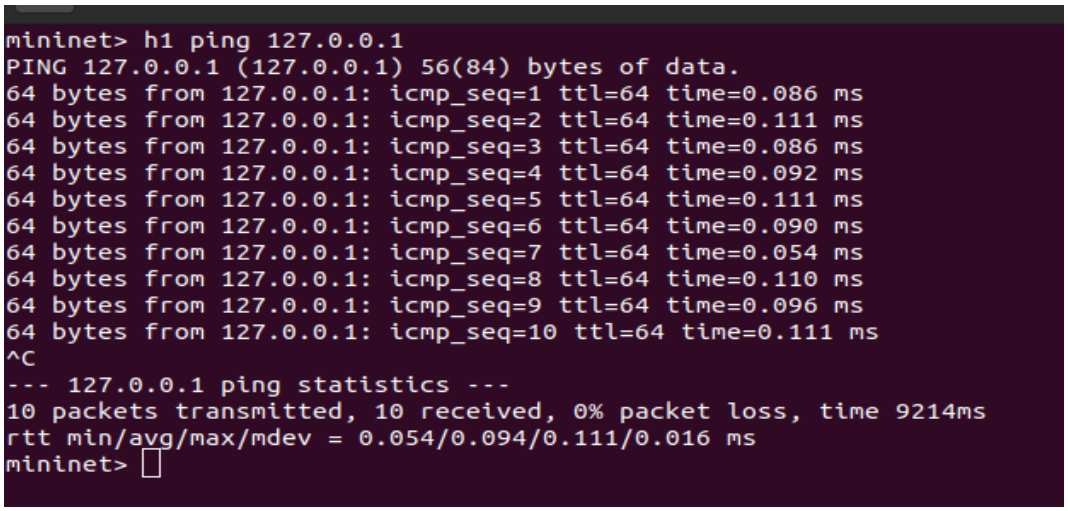


Figure 5.4: The ping command test from the host (h1) to the primary controller C_0 in Mininet

The scenario in Figure 5.4, assumes that the network topology is controlled by four controllers with integrated SA the primary controller is up and the networking is working in a perfect state as shown above. This same procedure was applied to all the secondary controllers (C_1, C_2, C_3..... C_{10}) in the network to check their performance using their respective IP addresses, as shown in Figure 5.5.

```
mininet> h1 ping 127.0.0.2
PING 127.0.0.2 (127.0.0.2) 56(84) bytes of data.
64 bytes from 127.0.0.2: icmp_seq=1 ttl=64 time=0.087 ms
64 bytes from 127.0.0.2: icmp_seq=2 ttl=64 time=0.096 ms
64 bytes from 127.0.0.2: icmp_seq=3 ttl=64 time=0.101 ms
64 bytes from 127.0.0.2: icmp_seq=4 ttl=64 time=0.113 ms
64 bytes from 127.0.0.2: icmp_seq=5 ttl=64 time=0.110 ms
64 bytes from 127.0.0.2: icmp_seq=6 ttl=64 time=0.110 ms
64 bytes from 127.0.0.2: icmp_seq=7 ttl=64 time=0.103 ms
64 bytes from 127.0.0.2: icmp_seq=8 ttl=64 time=0.086 ms
64 bytes from 127.0.0.2: icmp_seq=9 ttl=64 time=0.087 ms
64 bytes from 127.0.0.2: icmp_seq=10 ttl=64 time=0.097 ms
^C
--- 127.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9219ms
rtt min/avg/max/mdev = 0.086/0.099/0.113/0.009 ms
mininet>
```

Figure 5.5: The ping command test from the host (h1) to the secondary controller C_2 in Mininet

Using the same scenario in Figure 5.3, we assumed that the primary controller C_1 with IP address 127.0.0.1, goes down due to some network fault, error, or failure. This scenario is illustrated in Figure 5.6.

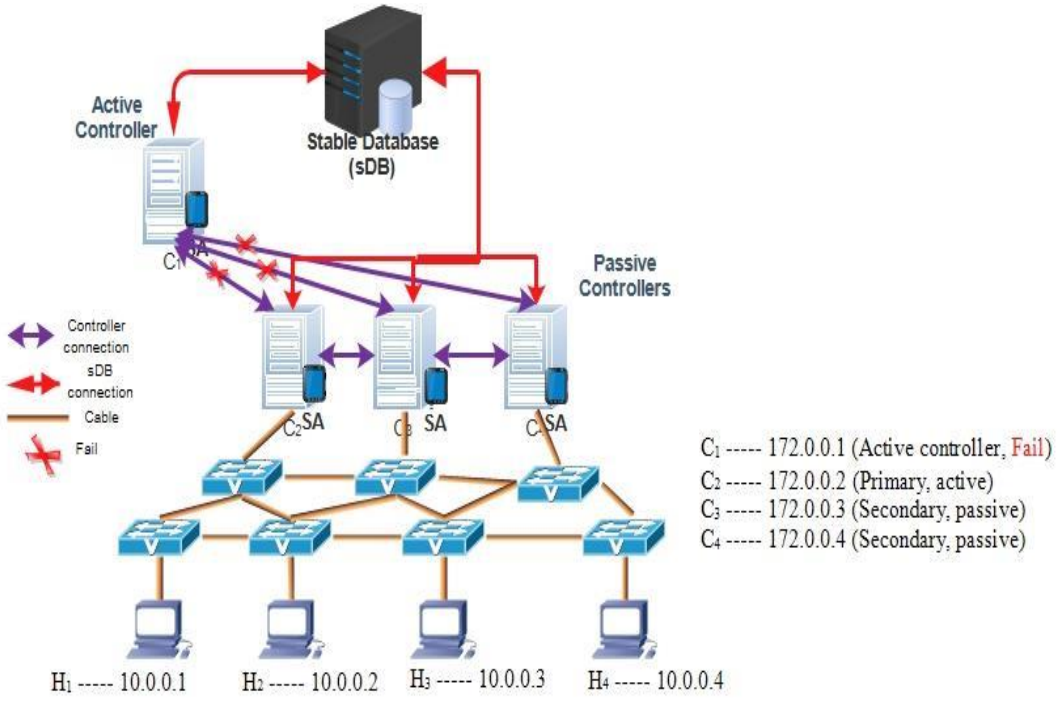


Figure: 5.6: Primary controller failure

In this case, there is a controller fault that occurs when the controller does not receive incoming demands from the switches or fails to send messages to respond to the switches' demand or an

error caused by the node/link that occurs. The SA collected network statistics requests to determine the type of fault or the error, whether it affects the hardware or link. Using the OpenFlow protocol the SA established an exchange message between them at intervals of 40 seconds, to detect the status of each controller on the network. If the SA in the primary controller sends a hello request and receives the reply in the interval of 40s, that indicates the controller is ALIVE; and if the backup/secondary controller does not respond in the interval of 40s, then the controller is assumed dead. In case a controller receives an incorrect message in response to the switches' message, the network cannot result in total failure. Using the above scenario C_1 was playing the role of the primary in all Ryu controllers' set-up topology, after a fault or a controller failure, the other controllers (secondary) elect one of them to take over as a new primary. This process is done automatically using the control programmable capability in Mininet. By using the *ovs-vsctl* command tools in Mininet the designation of the new controller was done.

Piece of code 4: Mininet Script to change the Primary Controller

```
$ ovs-vsctl set-controller C1 "tcp: 127.0.0.2: 6633"
$ ovs-vsctl set-controller C2 "tcp: 127.0.0.2: 6633"
$ ovs-vsctl set-controller C3 "tcp: 127.0.0.2  6633"
$ ovs-vsctl set-controller C4 "tcp: 127.0.0.2: 6633"
```

In this scenario, each controller gets the address of the newly elected primary controller and shares the information about the network and the same information is stored in the sDB. Referring to a scenario in Figure 5.5, after C_1 is dead, C_2 takes over the network immediately as the primary controller by using a request message, initiated by the OpenFlow to enable the controller to request a switch for port statistics using the *Ryu.ofproto.ofproto_v1_3_OFPPortStatsRequest* command in Mininet. Once the selection of the new primary is completed, new information is updated and sent to all the nodes in the network; then the details of the new primary are stored in the sDB. The primary controller is denoted by C_1 and any other controller C_n is secondary. For the simulation in Mininet, we make use of hosts as H_1 up to H*n* in the application plane layer as devices connected to the switches. The below piece of code shows the new network topology in the first scenario with all controllers after the election of the new primary using Python code in Mininet.

Piece of code 5: Controller topology after the election of the new primary in Mininet

```
"controllers": [
            {
          "opts": {
             "controllerProtocol": "TCP",
             "controllerType": "ref",
             "hostname": "c1",
             "remoteIP": "127.0.0.1",
             "remotePort": 6633
             "switchStatus":" DOWN",                    //Controller DOWN
          },
          "x": "405.0",
          "y": "290.0"
         {
          "opts": {
             "controllerProtocol": "TCP",
             "controllerType": "ref",
             "hostname": "c2",                          //New controller assigned
             "remoteIP": "127.0.0.2",
             "remotePort": 6633
             "switchStatus":" UP",
          },
          "x": "350.0",
          "y": "170.0"
       },
       {
          "opts": {
             "controllerProtocol": "TCP",
             "controllerType": "ref",
             "hostname": "c2",
             "remoteIP": "127.0.0.3",                   //Secondary controller
             "remotePort": 6633
             "switchStatus":" UP",
          },
          "x": "210.0",
          "y": "215.0"
            ……                                        //Continuity of the code
            ……                                        //Continuity of the code
       }
```

Using the *ping* test command above piece of code in Mininet from the host machine to the virtual switches, in the data path informed the controller of the packet's arrival that plays the role of the new primary, the application running on the new primary controller responded by identifying the updated flow tables' information from the switches and the communication was established between the nodes in the network.

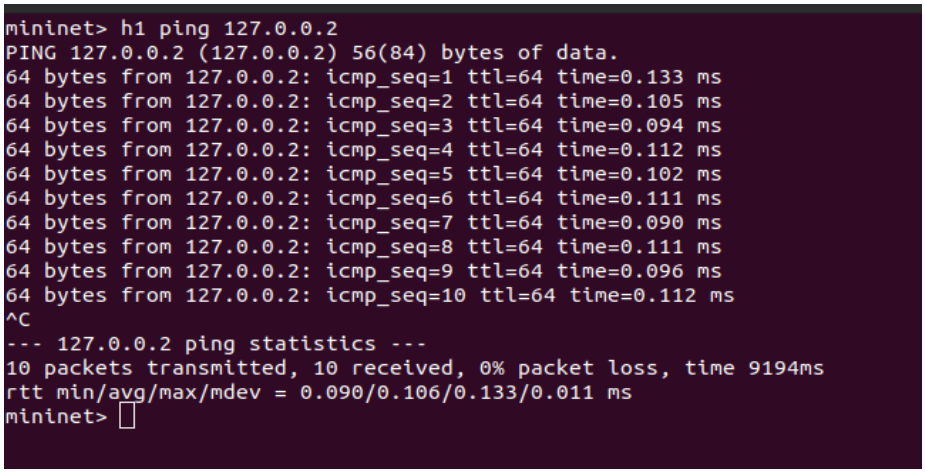


Figure 5.7: The ping command test after the election of the new primary from Host (h1) to C_2

Once the new topology is created, the virtual switches update the new flow table containing new information about the network state. The new details about the number of packets transmitted, received, and lost, as well as the new time, were updated and stored in the sDB.

5.6.2 Simulation results

The outcomes of the proposed FTM's simulated evaluation in a small-scale network scenario are shown in this section. As indicated in the previous section the evaluation in the small network makes use of modified controllers with SA. The controllers were evaluated by the latency and throughput performance. To measure latency and throughput performance for the SDN-FTF in Mininet, we make use of the standard topology presented in Figure 5.6, which is based on a small-scale network scenario. The controller running on the primary switch reacted to packet-in messages from the hosts connected to the virtual switches. After evaluating the performance of the controllers (Ryu and OpenFlow), Table 5.5 is the summary in terms of the runtime and round-trip time that determine the controller performances during the evaluation of the FTM.

Table 5.5: Controller details

Controller ID	Throughput (m/s)	Latency (m/s)	Type of controller
c1	-	-	OpenFlow
c2	1.066	9194	Ryu
c3	0.985	9207	OpenFlow
c4	1.084	9214	Ryu

To have a clear picture of the controllers' performances during the simulation using small-scale network topology, Figure 5.8 displays the throughput and Figure 5.9 the latency performance results of the scenario using both Ryu and OpenFlow controllers.

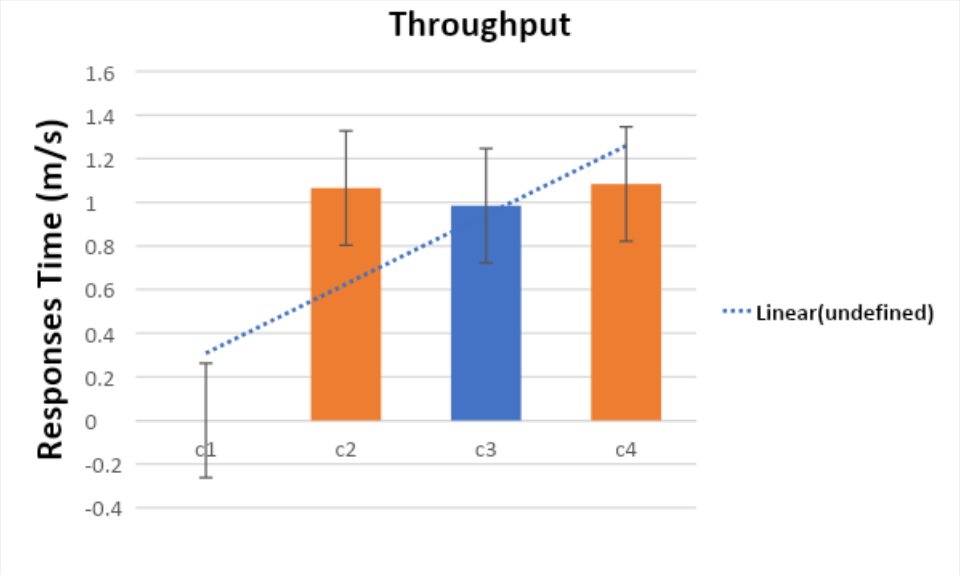


Figure 5.8: Individual controller throughput analysis

From the above, the performance of every single controller during the simulation using the proposed FTM in a small-scale network is presented. The controllers denoted by c1 and c3 were configured as OpenFlow controllers and c2 and c4 as Ryu's controllers. From the outcomes above, the individual controllers' performances of OpenFlow suffer because of the overhead and placement changes due to the selection of the new primary controller after the fault has occurred. In the above analysis, the throughput response times for the c2 and c4 (Ryu's) controllers are higher than for the c2 (OpenFlow) controllers. This implies that during the takeover of the new primary controller, each controller suffers in their performance, either due to higher load balancing or lower load balancing that caused their throughput to be high or low. It can be noticed that the OpenFlow controller throughput analysis is lower than Ryu's controllers.

The one for c2 and c4 (Ryu's) controllers is lower than c2 (OpenFlow). This stage c1 is considered as a fault and does not have any connection in the network. The same phenomenon was also discovered during the evaluation of the latency for the SDN-FTM in Figure 5.9, the controller latency analysis.

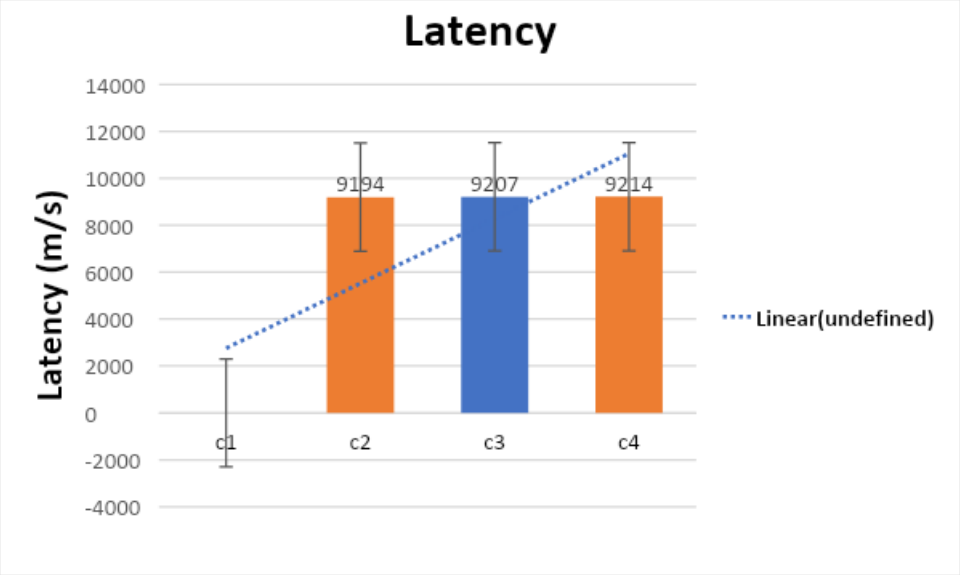


Figure 5.9: Individual controller latency analysis

However, the above latency analysis of individual controllers shows that the performances of c2 (Ryu) are less than c3 (OpenFlow), and c4 (Ryu) is better than c3 (OpenFlow). This suggests that each controller experienced performance issues during the selection of the new primary, nevertheless, their latency is not that much higher as compared to the throughput between the two. This implies that the network availability after the new primary is taken over is not delayed. After a fault was identified in c1, the SA listens to incoming requests using IP and port network statistics to identify the controller with less load balancing so that it can take responsibility for the primary controller. The SA's role is to listen on a certain SDN port at an IP address, while the other reaches out to the other to establish a connection of sorts. The one in the primary controller forms to the listener while the secondary reaches out to respond to the request.

The SA used within the context of this research is developed using a Python program and it runs in both controllers to maintain continuous communication with each other. It has three major methods, namely: the sender, the receiver, and the action of the SA. The SA designed in the context of this research is developed and implemented using a Python program language and it runs within both controllers to maintain continuous communication with each other. It has three major methods, namely: the sender, the receiver, and the action of the SA. As explained above previously, the deployment of the sender is shown in Figure 5.10.

```
1.  # Important Socket module
2.  import socket
3.  import sys
4.  # Creation of the SA using socket and the initiation of the "hello" message
5.  Try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            print ("HELLO message successfully sent")
6.  except socket.error as err:
            print ("HELLO message failed with error %s" %(err))
7.  # Default port for the controller housed the SA
8.  Port = 6633
9.  # Connecting to the SA housed in the secondary
10. Try:
            cntr_ip = socket.getcntrbyid("secondary controller")
11. except   socket. gaierror:
12. # Could not connect to the secondary
            Print ("there was an error resolving the secondary controller")
            sys.exit()
13. # Connect to the primary controller
14. s.connect((cntr_ip, port))
15. print ("the controller has successfully connected to the primary")
```

Figure 5:10: SA sender deployment program

To enable the establishment of SA, we began by importing the socket library and creating a basic socket. We passed two parameters after the socket instance was created. AF_NT and SOCK_STREAM are the first and second parameters, respectively. Within the context of this study, the address-family ipv4 that we employ as the IP range is referred to as AF_INET. The use of SO CK_STREAM. AF_INET shows that we are making the connection using the internet. Below are the steps:

Step 1: the creation of the SA (socket)

Step 2: resolve the controller IP address

Step 3: connect to the controller.

Step 4: identifying how to send the information through the SA

Step 5: Use the *sendall function* in the Python library to send information

Step 6: The information is sent to the SA (socket) housed in (a secondary controller) that is connected to.

Step 7: The SA housed in the secondary controller can send back the message using the same function.

In this process, the SA receiver can listen to incoming connections, accept them, and end the process. The SA initiates a connection with other SAs housed in the controller. For the SA to be able to listen, there is a need for the receiving method that allows incoming requests using IP in the SDN port to be received. The deployment of the receiver is shown in Figure 5.11.

```
1.  # Import all the socket library
2.  import socket
3.  # Creation of the SA (socket) object
4.  s = socket.socket()
5.  print ("HELLO message successfully sent")
6.  # reserve a controller port in our case its 6635
7.  Port = 6635
8.  # Next bind to the port
9.  s.bind(('',port))              # No IP is specified, the controller
10. print ("SA binded to %s" %(port))
11. #Put the SA into listen mode for all the controllers in the network
12. s.listen(3)
13. print ("SA is listening")
14. # Making use of a loop until there is an interruption or an error
15. While True:
16. # Establish a connection with the SA housed in the secondary controller
17. C, addr = s.accept()
18. Print ('received a connection from' addr)
19. # Send a receiving request to the SA house in the primary controller
20. c.send('message received, and communication granted' ,encode())
21. # Close the communication
22. s.close()
23. #Breaking the connection with the controller
24. break
```

Figure 5.11: SA receiver deployment

From the receiver method piece of code above, we import all sockets because it is required to create our SA object. After that, we create the object (SA) using the socket by setting a port aside for it. Following is that the primary controller that houses the SA_p is designated using the IP address and the port. When the SA housed in the primary has an empty message, it can also accept inbound connections from other controllers on the network. The SA would have only listened to requests made by one control if we had passed the 127.0.0.1 IP address. However, we set the primary controller on the listen mode by indicating the number 3, which means that the primary controller waited for three connections to listen from. In case the 4^{th} fourth connection tried to connect, the connection was declined and there was an error message. Finally, we create a while loop, begin accepting all incoming connections, and then begin to close those connections after sending a receiver thank you message to each connected socket.
For the SA to operate and work in its environment as well as in the controller that housed it, we created a piece in Python for that functionality as shown in Figure 5.12.

```
# Important Socket module
import socket
#Importing the socket library and making a simple socket
s = socket.socket(socket.AF_INET, sockets.SOCK_STREAM)
# Create a socket object in this case stationary agent (SA)
s = socket.socket()
# Define the SDN port on which the connection needs to be made
Port = 6633
# Connect to the SA housed in the secondary Controller Cs
s.connect (('127.0.0.1'))
# Receive information from the SA house in the primary Controller Cp and decode to get
the message
print (s.recv(1024).decode())
# Close the connection
s.close()
```

Figure 5.12: SA action

The above piece of code presents the structure of the SA operation. Firstly, we import the socket library and make a simple socket that allows the creation of the object (SA). Next, the SA establishes a connection between the controller by defining the SDN port on which the connection needs to be made, after which it requests the information from the SA housed in the secondary and then cuts off the connection. To see the latency and throughput performance of the SDN-FTF, we have used Wireshark [68][106]. Wireshark is a Mininet tool that is used to evaluate controllers' and switches' communication to determine how packets pass through the switches to their destinations [68][106]. It can also be used to evaluate the latency and throughput performance of the controllers. Based on these results, we performed a general network performance evaluation looking at the throughput analysis response time as well as the latency analysis for each controller to determine the performance-based throughput and latency. Table 5.6 shows the values of controller performance in terms of response time (m/s) and Latency (m/s).

Table 5.6: Controller Performance Comparison

Type controller	**Throughput (m/s)**	**Latency (m/s)**
OpenFlow	2.15	18408
Ryu	0.985	9207

The above summary result of the general network performance evaluation for the throughput and the latency analysis comparison based on small-scale network topology to identify the best topology performance for the proposed SDN-FTM from the result is displayed in Figure 5:13 and Figure 5: 14.

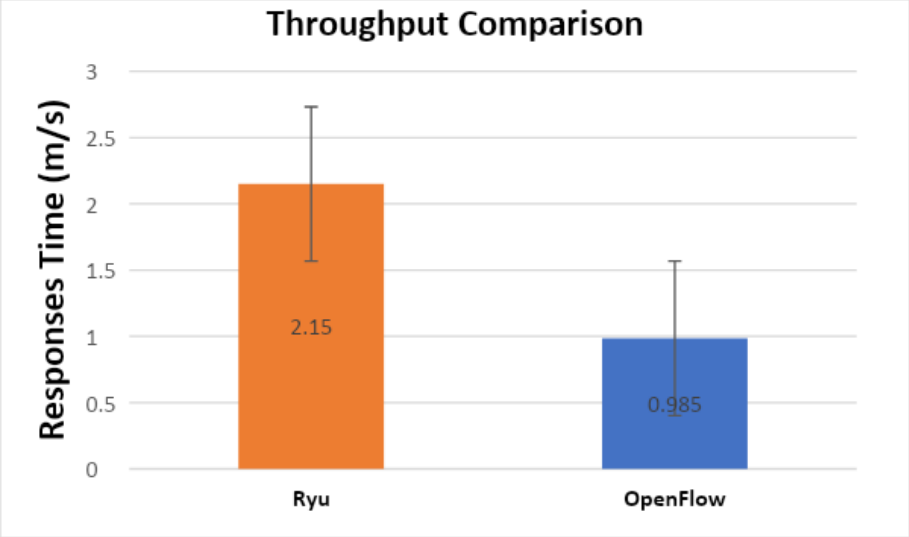


Figure 5.13: Controller throughput comparison

Following the above, it is revealed that Ryu's controllers have an increased throughput performance compared to the OpenFlow controllers. It is evident that from the experiences, the general network throughput analysis for the proposed SDN-FTM for a small scall network has the best performance results and shows the best scalability compared to the others. It shows that Ryu's controllers for the proposed SDN-FTM performance have a throughput analysis twice as high as OpenFlow controllers. This indicates that the quantity of data sent in the network small scale for the proposed SDN-FTM is being processed faster in a specific period during fault-tolerance compared to OpenFlow controllers. This higher rate of throughput shows that messages sent on the network arrive at the destination successfully and are being delivered with less delay time. These results are indifferent to the low rate of throughput shown in the OpenFlow controllers. Likewise, the general network performance evaluation for the latency analysis for the proposed SDN-FTM is presented in Figure 5.14.

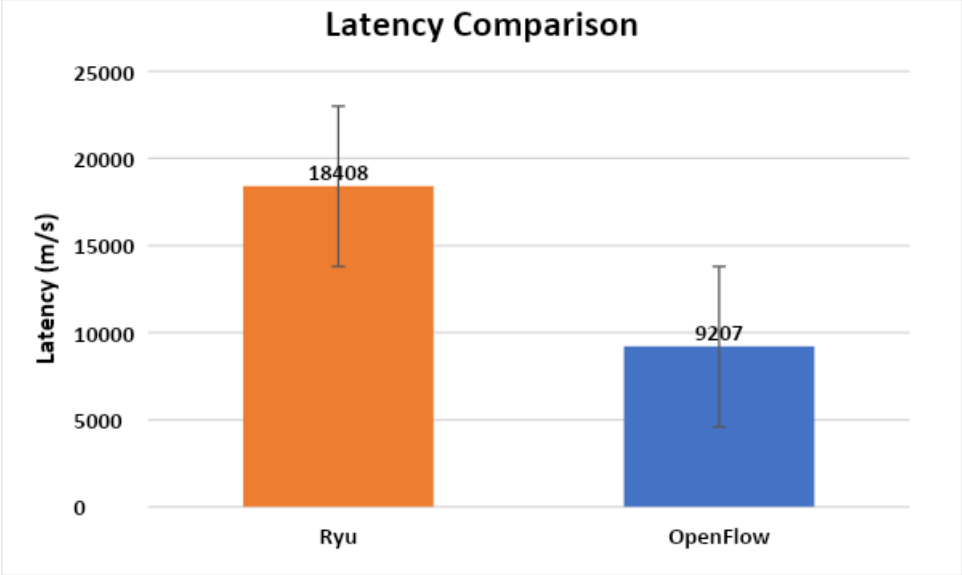


Figure 5.14: Controller latency comparison

The above results show that the OpenFlow controllers have a low latency rate compared to Ryu's controllers. This low latency demonstrates that in a small-scale network for the proposed SDN-FTM, OpenFlow controllers' responses after failure are better than Ryu's. Their latency and throughput performance are not the same in the small-scale network scenario. The following chapter evaluated the proposed FTM in a medium and large-scale network. The controllers used in the scenario are normal controllers without any modification and the results were compared to see how best the FTM performs.

For the proposed SDN-FTM, latency and throughput are crucial performance metrics to consider. To evaluate the efficacy and efficiency of the proposed SDN-FTM, they must be justified in the context of this study. Since latency and throughput have a direct impact on fault detection speed, fault recovery time, network resilience, scalability, and overall network performance, their use as performance indices for the proposed SDN-FTM is explained. In the experiments, low latency guarantees that faults are detected and recovered quickly, and high throughput ensures that recovery traffic is handled effectively without impairing normal network operations. The proposed SDN-FTM can offer dependable and effective fault tolerance in software-defined network environments by optimising both throughput and latency.

5.7 Summary

This chapter presents the evaluation, and simulation of the proposed SDN-fault tolerance model (FTM). After the design in the previous chapter, the proposed SND-FTM was developed and evaluated in real-time in small-scale network scenarios using different parameters in the Mininet simulation tools. The results from the experiments were discussed in detail using various performance measures such as latency and throughput. The outcomes show that the proposed SDN-FTM using the modified controller with SA, had better latency performance using the Ryu controller than OpenFlow in small-scale network scenarios analysis. In addition, a competitive analysis of the results was made and discussed, it was shown that the proposed SDN-FTM can offer reliable and effective fault tolerance in small network sizes.

Chapter 6

FTM Implementation and Evaluation in Medium-to-Large-Scale Network

6.1 Introduction

A major difficulty for SDN with distributed controller architecture is the CCP, deciding on the optimal location and the right number of controllers to deploy is this CPP's objective to maximise network performance. For the proposed design, we have taken on the CPP that accounts for the distribution of the controller workload, control plane utilisation, and network communication delay.

This section presents the development and evaluation of the SDNFT controller placement to ensure that the number of controllers that need to be deployed is placed in a good position. Using various distance matric and parameters, we employ the k-median and k-means algorithms discussed in Chapter 4, using MATLAB for the simulation and deployment.

6.1.1 Outline

The issue of controller placement is addressed in this chapter, as well as the fault tolerance simulation for the proposed SDN-FTM, presented in the previous chapter. The chapter is divided into two sections: the first part presents the controller placement implementation and simulation. The second section presents the SDN-FTM evaluation and simulation without a modified controller and finally compares the analysis is compared the one in the previous chapter.

6.2 Controller placement

This section presents the proposed controller placement technique in this research.

6.2.1 Overview

This section presents and assesses the controller placement issue. The number of controllers and switches that must be put in the network, as well as their placement, are calculated using mathematical models to formulate the problem. This study aims to identify the controller's ideal position to decrease average controller latency, boost dependability, and guarantee high resource availability in the network. To prevent resource and cost, it is necessary to specify the number of controllers required for the network, and where should they be located. Based on a

comparative analysis of SDN CPP carried out in Chapter 2, specific techniques are used to choose the optimal location for the proposed FTM. By choosing the best location for the controller and minimising cost, distributed SDN controllers can provide fault tolerance in contrast to centralised SDNs. The number of SDN controllers needed their location, and how to reduce latency between controllers and between controllers and switches to improve network performance and save costs all depend on placement [54].

As characteristics on which the needs to define the problem are based, the theoretical analysis in Chapter 2, considered the number of controllers, latency, and position area of the controllers during the deployment. Based on the foregoing, UCPP was selected as an appropriate technique for placement for the proposed FTM due to its limitless capacity and lack of knowledge of controllers' loads [54]. Therefore, to design UCPP to be employed for the FTM, the minimum k-centre and minimum k-media algorithms were selected and deployed [61]. This is shown in equation (6.1).

$$Placement = (Rbty) \quad \text{.....(6.1) [61]}$$

For the evaluation of the controller place problem, we defined the system properties and evaluation parameters needed for the simulation. A multiple-controller approach architecture segment in small to medium-manageable clusters is designed to evaluate controller placement in a small-scale network with topology and metrics. The anticipated results of the evaluation are determining the optimal controller placement distance between several switches in the clusters, the capacity of the controller increased, and the average propagation latency decreased when the ideal location for those controllers is determined.

6.2.2 Problem formulation

The SDN-enabled network can be used to formulate the network partition of the SDN problem, with the nodes and links of the physical topology of undirected graphs given as in equation (6.2).

$$G = (V, E) \quad \text{.....(6.2) [61]}$$

Where E is the set of physical links connecting the nodes, and V is the SDN set of nodes that represents the network Switches and controllers. Instead, the network partition can be described

as in equation (6.3) if the number of partitions is given as k when SDN is taken into consideration.

$$SDN_i(V_i, E_i) \quad \text{……..(6.3)}$$

Equation (6.3) is subject to conditions in equations (6.4) to (6.7).

$$\cup_{i=1}^{k} V_i = V; \ \cup_{i=1}^{k} E_i = E \quad \text{……..(6.4)}$$

The equation (5.3) Is used to denote the total number of subnetworks required to cover all network nodes and linkages.

$$SDN_i \cap SDN_j = \emptyset; \ \forall_{i \neq j, \ i,j \in k} \quad \text{……..(6.5)}$$

In this case, ∅ is denoted Is used to denote the total number of subnetworks necessary to cover all network components, including nodes and links.

$$similarity(SDN_i) = TRUE, \ \forall_{i \neq k} \quad \text{……(6.6) [61]}$$

Equation (6.5) indicates that the elements in one sub-network have the same similarity.

$$similarity(SDN_i \cap SDN_j) = FALSE, \ \forall_{i \neq j, i, j \in k} \quad \text{……(6.7) [61]}$$

The elements assigned to various subnetworks appear to have s=distinct properties, according to equation (6.6), when the network is anticipated to be partitioned so that the nodes in one cluster have a smaller latency equation (6.6), while the nodes in separate clusters have a bigger latency equation, this is when the similarity is defined as the latency equation (6.7).

SDN_I are connected regions ……(6.8) [61]

All of the vertexes in one sub-network are linked together, according to equation (6.7). The network is divided into smaller sub-networks using the k-means clustering-based partition algorithm to reduce the maximum end-to-end distance. To be clear, the initial nodes chosen to

run the clustering algorithms are referred to as the “centre”, the true centre, or “centroid”, of each cluster. The following part describes the evaluation and simulation of the controller placement using k-media and k-means algorithms discussed in Chapter 4 section 4.4.1 to determine the total distances between the cluster’s centre and every other point for controller deployment.

6.3 Evaluation process

This section presents the evaluation of the proposed SDN-FTM based on placement and fault tolerance based on various system properties, evaluation metrics and simulation parameters. In addition, various steps used for the simulation are presented as well as the simulation results.

6.3.1 System properties

The properties presented in Chapter 5 section 5.3.1 of the simulation set-up, are being used in this section to perform the simulation of the proposed system.

6.3.2 Methodology

This section represents the design of the experimental steps used during the simulation within the context of this study. As described in the literature review (Chapter 2) and research methodology and design (Chapter 3), various approaches can be used in computer science research to validate a design, within this context simulation was suitable for proof of numerous assumptions in this research [58]. From a practical aspect, it is important to note that all the simulation experiments in this study are implemented on computers using various software applications appropriate for this kind of research. Once suitable software for performing experiments was identified, a couple of steps were developed to make the simulation easy within this context. Figure 6.1 summarises these steps.

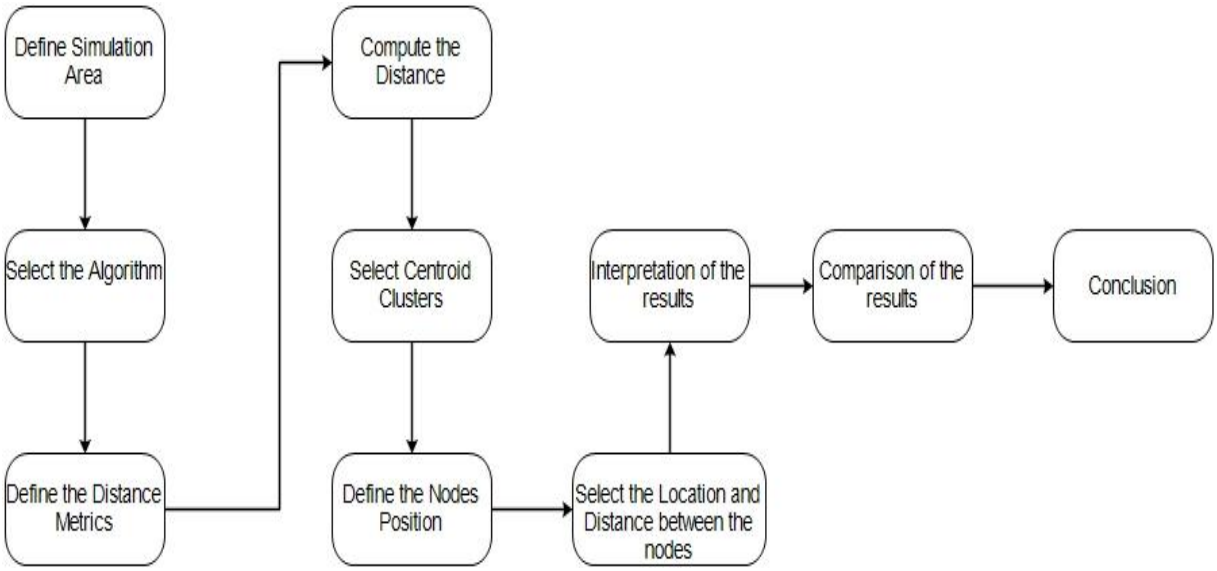


Figure 6.1: A comprehensive simulation steps

To address the issue of controllers' placement in the proposed SDN-FTF architecture, we employ the *city-block* distance metric in MATLAB using equations (4.1) and (4.2) in Chapter 4 in section 4.2.3, to determine the optimal position for the controllers in network infrastructures while minimising the maximum distance, as well as to calculate the shortest link on the network that minimises the average propagation delay between the network nodes [96]. When using *Cityblock* distance (also known as Manhattan distance) to calculate the sum of absolute difference *x* plus the distance in *y*, we deployed K-means and K-medoids clustering to solve this problem. In the *Cityblock* distance metric, each centroid is the component median of the point in that cluster. This is similar to defining the distance between points, cities, and areas (like in Manhattan) and how to move around streets and buildings instead of moving directly to one location [96]. The *Cityblock* the separation between two points, *a* and *b* with *k* dimension is denoted and calculated using Equation (6.11) as follows:

$$\sum_{j=i}^{k} | aj - bj | \qquad \ldots\ldots\ldots\ldots\ldots \qquad (6.11)\ [96]$$

To compute the distance using different distance metrics and to find out the centroid clusters between the controllers and switches whereby *x* represents the distance and *c* the centroid, the equation below is implemented in Equation (6.12):

$$d(x,c) = \sum_{j=i}^{k} | aj - bj | \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (6.12)\ [96]$$

The main objective of using the above equation is to compute the k-means clustering algorithm to define the location of the controllers as well as specify the distance between them.

6.4. Controller location and allocation

Based on the above procedure and using equation (6.12) we performed a simulation in MATLAB to achieve our goals of defining the best places for the controllers to be installed and maximising the number of controllers to be deployed in the proposed SDN-FTM network infrastructure [96]. To address the above goals, we made use of K-mean clustering and K-medoid clustering. To assess this, we created a set of random numbers using *x, and y* coordinates and made use of a matrix that holds randomly generated usage numbers for each switch and defines the number of clusters to be used (essentially = to the number of controllers) on the network. To exhibit the planned network topology partition, as shown in Figure 6.2, the distance between switches and controllers must be specified as a fundamental component of this mathematical model.

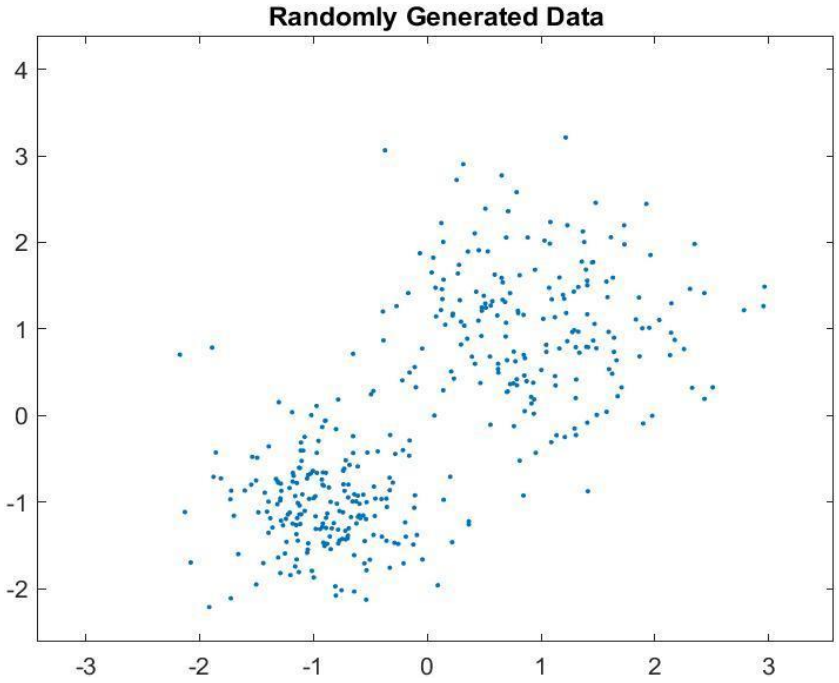


Figure 6.2: Placement topology formation

The plot shown in Figure 6.2 displays the network partition randomly generated to define the best arrangement of switches. This partition is crucial for choosing the right number and placement of controllers to deploy. In this scenario, the locations are generated randomly.

For the deployment of multiple controllers, we segment the network topology into medium to large-scale manageable clusters to overcome the challenge of the single controller by initialising the *city-block* distance metric randomly using the K-means algorithm as shown in Table 6.4 to determine the optimal controller placement distance between several given switches in the clusters.

Table 6.4: Distance metric

Replicate	**Iterations**	**Total Sum of Distances**
1	11	88.2602
2	8	92.3789
3	9	89.3037
4	6	92.6302
5	8	91.1183
6	7	96.1411
7	10	92.0779
8	9	93.9656
9	6	91.2769
10	7	99.2522
Best total sum of distances		**88.2602**

Out of the 10 randomly initialised instances of the K-mean, with various iterations, the minimum sum of the distance between all points clusters switches and their centroid is 88.2602. Considering this distance matrix, we plot the cluster allocations that define the best network sections using the above matrix to partition them into 10 small segments (clusters) as shown in Figure 6.3. Each segment is represented with a specific colour to allow the plotting of the centroids for the controllers' deployment. The number of clusters generated is essentially equal to the number of controllers needed to be deployed.

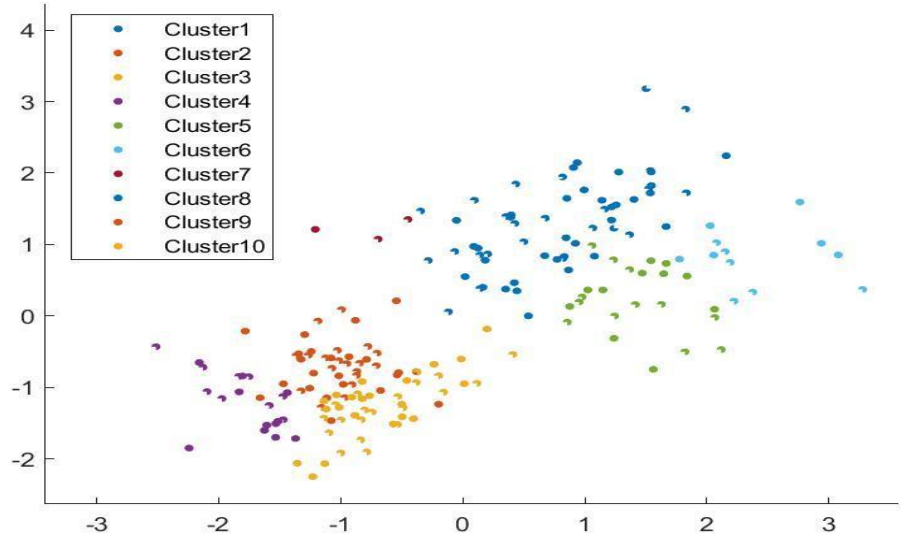


Figure: 6.3: Cluster allocations

Reducing the average controller propagation latency is the primary goal of solving the CPP, increases reliability, and always ensures availability in the network infrastructures. Analysing various distance metrics after each iteration in Table 6.4 reveals that the distance between the switches and controllers is minimised. This shows that deploying more controllers increases the capacity of the controller and decreases the average propagation latency between the clusters in the network architecture. See Figure 6.4.

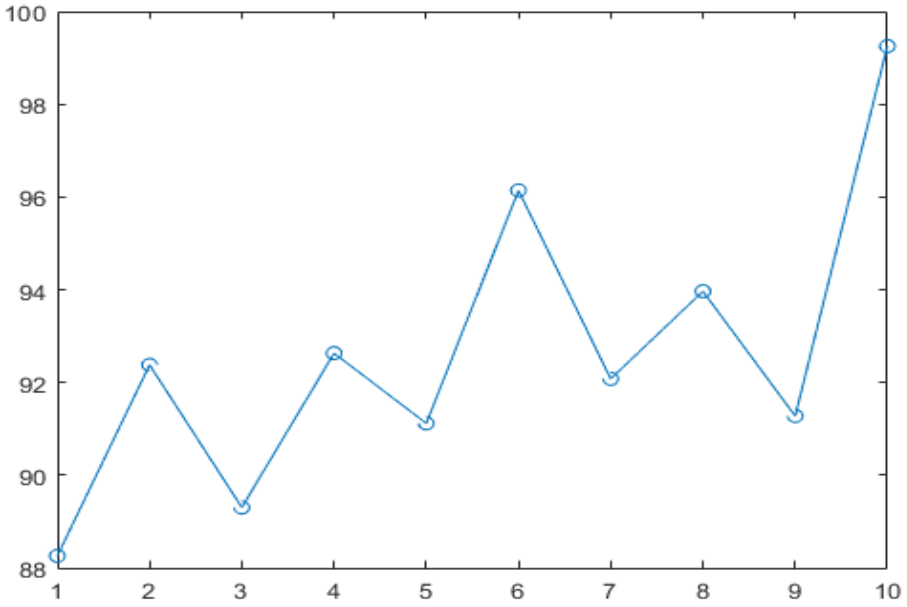


Figure 6.4: Propagation latency between the clusters

To confirm the above statement, we used the same distance metric in Table 6.4, but we minimised the number of replicates as shown in Table 6.6. By relating the analysis in Figure

6.5 shows that the propagation delay between the controllers is very high compared to Figure 6.4.

Table 6.2: Distance metric with fewer replicates

Replicate	Iterations	The total sum of distances
1	11	88.2602
2	8	92.3789
3	9	89.3037
4	6	92.6302
5	8	91.1183
Best total sum of distances		88.2602

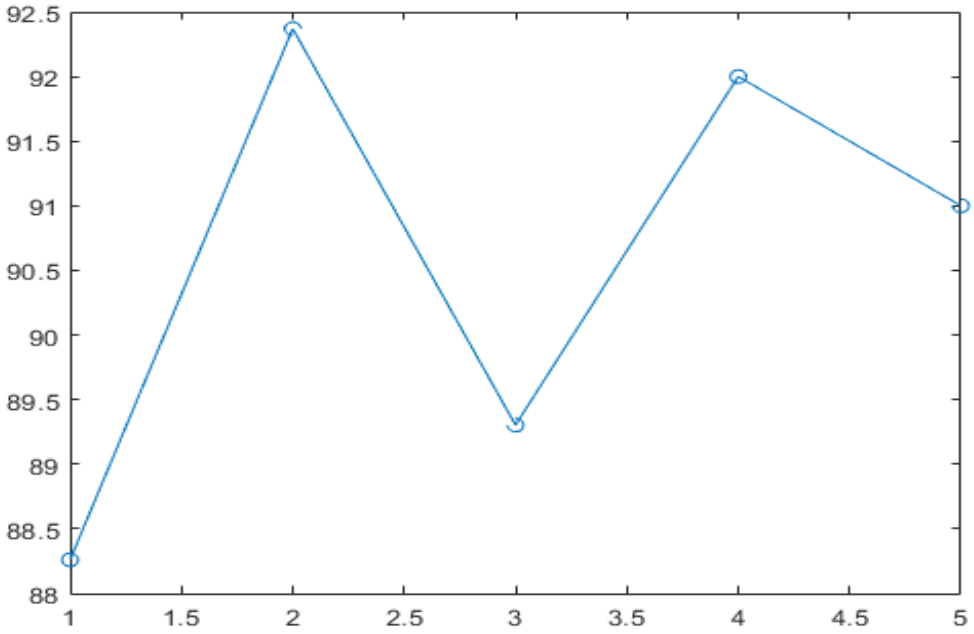


Figure 6.5: Propagation latency with fewer replicates

According to the literature, the switches-controller distance is a key factor that imposes fundamental limitations on the controller availability and propagation delay in the network. Looking at both analyses it is evident that, with fewer controllers deployed the capacity of the controller to handle traffic is compromised and this may increase the balanced loading of each controller in the network. However, the more controllers deployed, the more it increases the capacity of the controllers, the reliability of the network, and always ensures the availability of resources.

After obtaining the best distance metric between switches and controllers, we must specify the best location for their placement. We make use of *K-medoid* clustering algorithms because of

their robustness to plot the centre of each cluster as the central location with a minimum sum of distances to the switches to place the controllers [96]. The idea is to have a medoid as a central location that reduces network interference and delay to minimise the maximum distance metric of the switches to the closest controller C_n for better placement in the best location represented in x_n. See Figure 6.6.

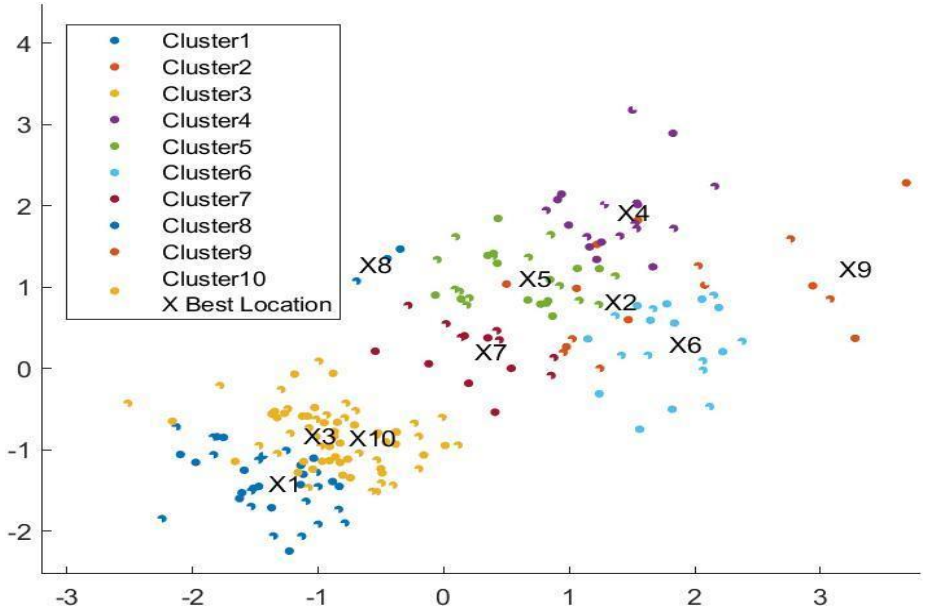


Figure 6.6: Best controller locations

Looking at Figure 6.6, we discovered that the best location for controller placement, in this case, is not fixed but was chosen randomly using the best metric as a central point of each cluster (network segment). The plotting of these best locations for placement (centroids), shows the precise spots in the network topology design that reduce average latency where the controllers need to be placed. The plot shown in Figure 6.7, demonstrates the actual controller location placements whereby the ideal number of controllers, in this case, is *Cn=10*, whereby C_1 represents controller 1, C_2 controller 2, and so on.

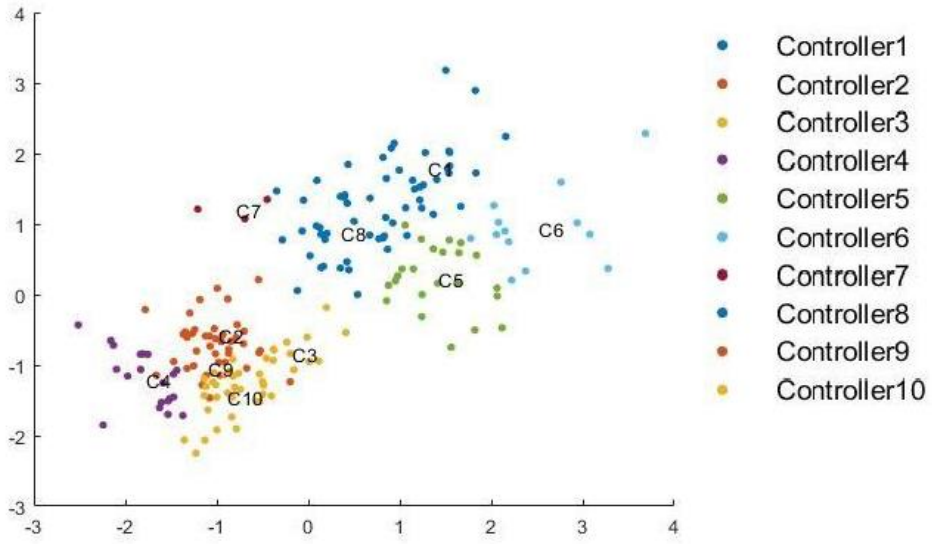


Figure 6.7: Controllers locations

Figure 6.7 displays the best SDN controller location deployment for the proposed SDN-FTF that guarantees network reliability, availability of resources always, as well as best network performance.

6.5 Evaluation of the proposed FTM based on controller placement.

As indicated in the previous section the deployment of controller placement addresses the issue of a single controller overseeing the network that creates a single point of failure in SDN. Utilising more than one controller (multiple controller approach) can improve network fault tolerance, scalability, load balancing, and QoS as well as network performance. In most cases, the deployment of a distributed control plane increases system availability and reliability, and security issues may also be improved [99]. Furthermore, this approach affects throughput and latency performance, and therefore, the location of the controller (placement) inside the network is crucial to the network's functionality. Unlike Chapter 5, this chapter focuses on the placement as well as the evaluation of the proposed SDN-FTM using multiple controllers in the design architecture presented in Chapter 4. In this experiment, we have added more controllers in the proposed SDN-FTM architecture in a distributed approach compared to the one presented in Chapter 5. Compared to the traditional SDN architecture, our proposed one can provide increased fault-tolerant features and improve independent controller management as well as throughput and latency.

Using the same OpenFlow SDN-FT Controllers Setup in Mininet as presented in the previous chapter, we applied the *city-block* distance metric values that were generated randomly using the K-means algorithm in Table 6.4 to define the "*x*" and "*y*" values during the experiment for the best optimal controller placement distance between several given controllers and switches in the clusters. Additionally, we have increased the number of controllers from four (4) in Chapter 5 to ten (10), and we have increased the IP addresses range from 172.0.0.1/10 while keeping the same number of switches and hosts. See Figure 6:8.

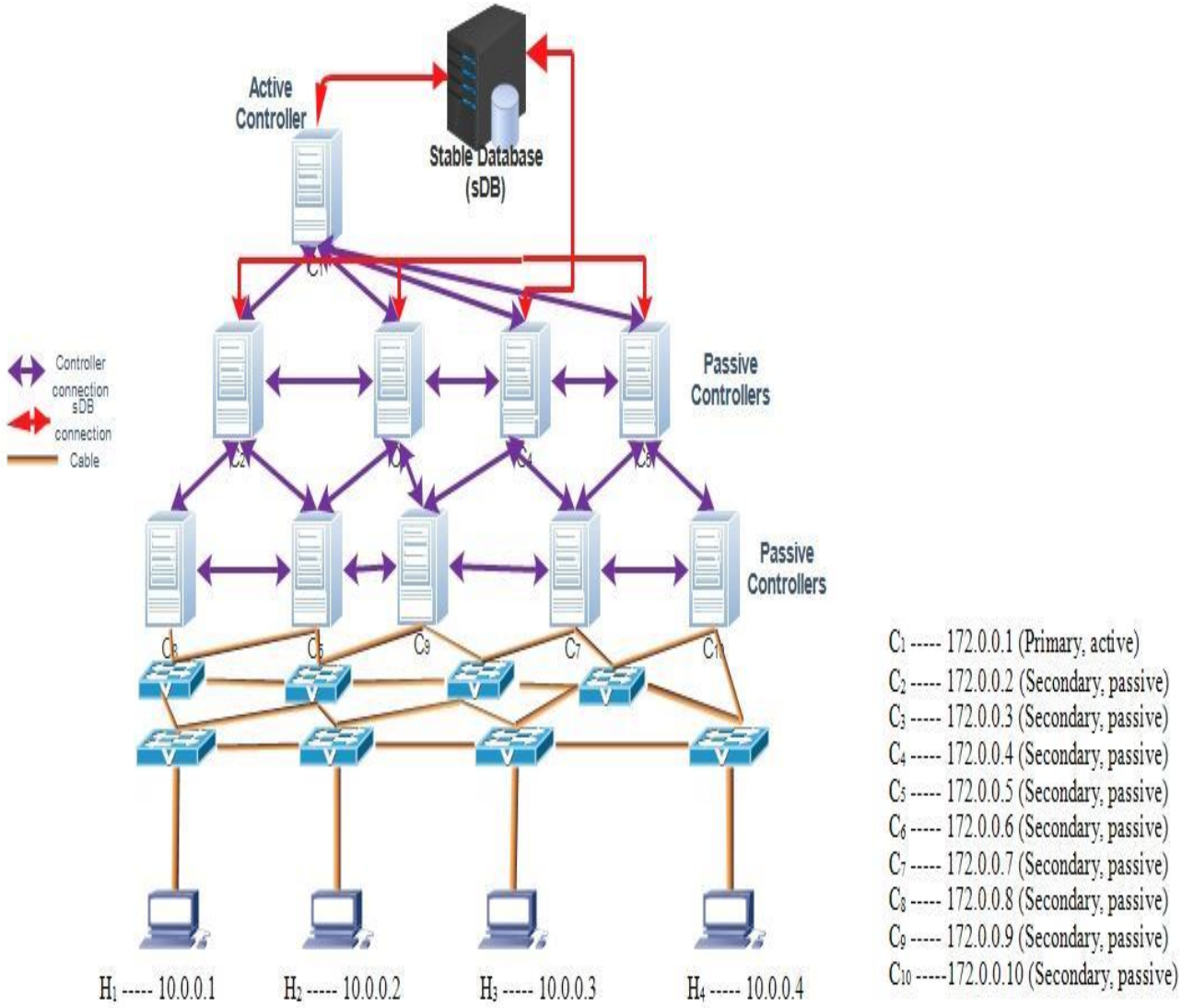


Figure 6.8: Architecture for medium to large-scale networks

Following the same procedure applied in the experiment on a small-scale network in the previous chapter, the controller c_0, in this case, is chosen as the primary in this initial

experiment, as shown in Figure 6.8 above, and is recognised and connected to all the other controllers and switches in the network in a distributed fashion. This type of connection enables synchronisation whereby node shares information about their status in the network, thereby removing the single point of failure that is experienced by a centralised controller. The scenario in Figure 6.8, assumes the deployment of five Ryu's controllers and five OpenFlow controllers; the primary controller (active) is an OpenFlow one denoted by c1 with Ip address 127.0.0.1 and the rest are the secondary controllers (passive).

6.6 Performance analysis

In the initial stage of the experiment, using the same procedure used in the previous chapter, in the initial stage, the switches' flow table is empty; once a fault occurs on the primary controller (c1, (OpenFlow)) with IP address 127.0.0.1, assuming the controller goes down due to some network fault, error, or failure. This scenario is illustrated in Figure 6.9.

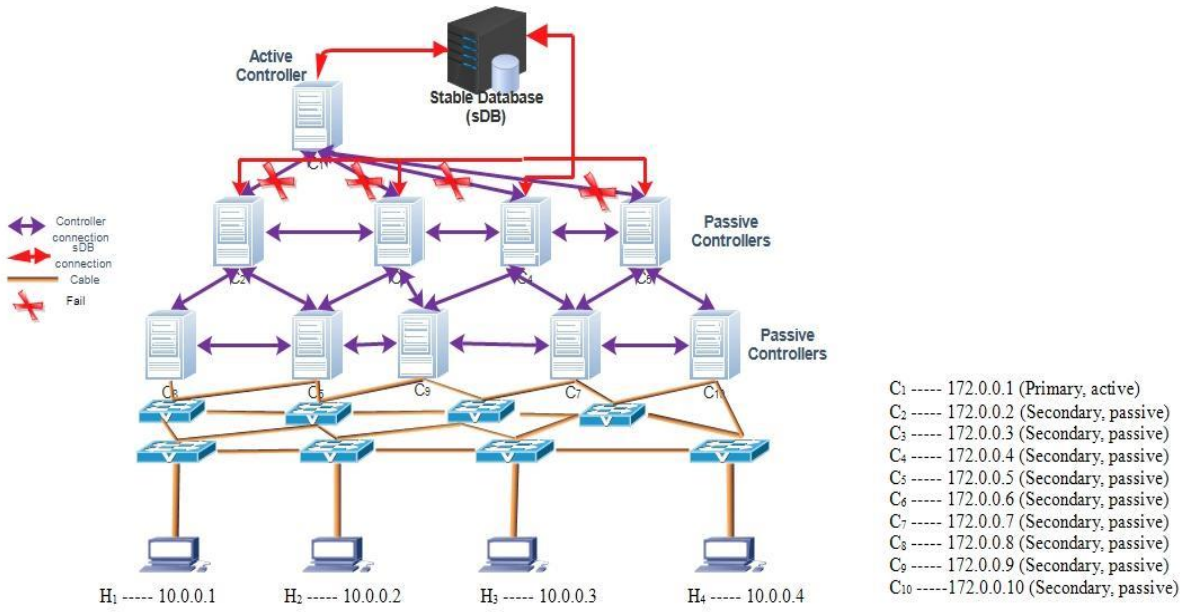


Figure 6.9: Controller failure in medium to large-scale networks

Using the *ovs-vsctl* command tools in Mininet as in the previous chapter, a new controller was selected individually. The controller gets the address of the newly elected primary and shares the updates. From the above figure, the failed state indicates that the controller has failed/crashed; on the other hand, the active state indicates that the controller is functioning. To provide a summary of the performance of the individual controller during the fault tolerance, we display the throughput in Table 6.5, using the runtime values of each controller which are expressed in m/s in Table 6.6.

Table 6.3: Controller throughput analysis

Controller	Type of Controller	Controller Protocol	Status		IP Address	Packet drop (%)	Run-time/ms
c1	OpenFlow	TCP	DOWN	DOWN	127.0.0.1	-	-
c2	Ryu	TCP	Active	UP	127.0.0.2	0	1.066
c3	OpenFlow	TCP	Passive	UP	127.0.0.3	0	1.01
c4	Ryu	TCP	Passive	UP	127.0.0.4	0	1.084
c5	OpenFlow	TCP	Passive	UP	127.0.0.5	0	1.032
c6	Ryu	TCP	Passive	UP	127.0.0.6	0	1.037
c7	OpenFlow	TCP	Passive	UP	127.0.0.7	0	1.013
c8	Ryu	TCP	Passive	UP	127.0.0.8	0	1.068
c9	OpenFlow	TCP	Passive	UP	127.0.0.9	0	0.986
c10	Ryu	TCP	Passive	UP	127.0.0.10	0	1.055

Using the round-trip time (RTT)/ms in Table 6.7, the researcher performed the round-trip time (RTT)/ms for Ryu and OpenFlow controllers for the proposed SDN-FTM to gain more understanding of the effect of latency during network fault or error for both controllers.

Table 6.4: Controller latency analysis

Controller	Type of Controller	Controller Protocol	Status		IP Address	Packet drop (%)	Round trip time (RTT)/ms
c1	OpenFlow	TCP	DOWN	DOWN	127.0.0.1	-	0
c2	Ryu	TCP	Active	UP	127.0.0.3	0	0.9001
c3	OpenFlow	TCP	Passive	UP	127.0.0.3	0	0.9398
c4	Ryu	TCP	Passive	UP	127.0.0.5	0	0.9011
c5	OpenFlow	TCP	Passive	UP	127.0.0.5	0	0.9379
c6	Ryu	TCP	Passive	UP	127.0.0.7	0	0.9101
c7	OpenFlow	TCP	Passive	UP	127.0.0.7	0	0.9388
c8	Ryu	TCP	Passive	UP	127.0.0.9	0	0.9001
c9	OpenFlow	TCP	Passive	UP	127.0.0.9	0	0.9398
c10	Ryu	TCP	Passive	UP	127.0.0.10	0	0.9012

6.6 Discussion

This section discusses the discuss the results of the experiments using throughput analysis and latency analysis of the proposed system and their implications in the context of this study.

From the results from the previous section, graphic representations were used to discuss the findings as well as to give a better visual illustration of the performance of the controller. To have a clear picture of the controllers' performances during the evaluation of the proposed SDN-FTM using the two scenarios during the simulation, the outcomes show that the performance of Ryu's controllers gives a better response time in case of a fault compared to the OpenFlow ones for every individual controller in the network. This shows that Ryu's throughput is higher than the OpenFlow, which implies that the Ryu's controllers' response time during taking over of the new primary controller is faster than the one for the OpenFlow controller. The experiment from the previous chapter and this are comparable, which assumes that controllers suffer in their performance. However, it is confirmed that in the proposed SDN-FTM, OpenFlow controllers suffer more greatly than Ryu's see Figure 6.10.

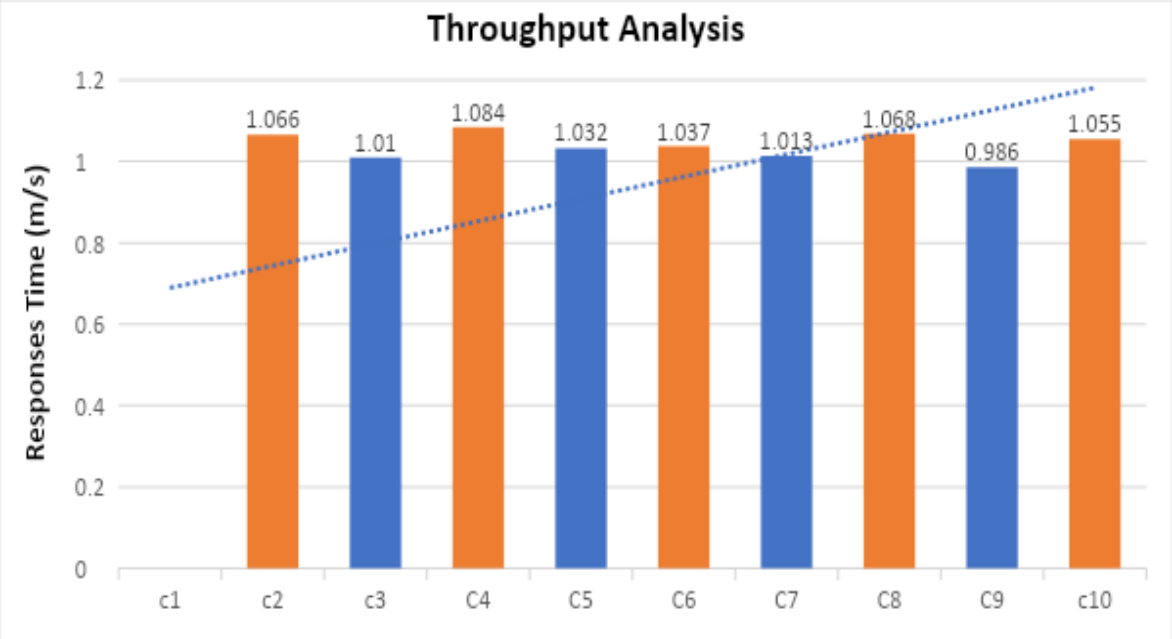


Figure 6.10: Controller throughput analysis

Unlike the previous experiment in Chapter 5, the results here are different. Ryu's controller for the proposed SDN-FTM in distributed deployment when using placement and an increased number of controllers shows a low latency rate compared to the experiment in Chapter 5. This implies that the performance evaluation for the latency analysis shows better performance of the system compared to the OpenFlow controllers. These results may be due to the use of

multiple controllers and the selection of the best placement. This allowed the proposed SDN-FTF to increase its performance as well as the sharing of packets sent among controllers during the selection of the primary in the event of a fault. The illustration is in Figure 6.11.

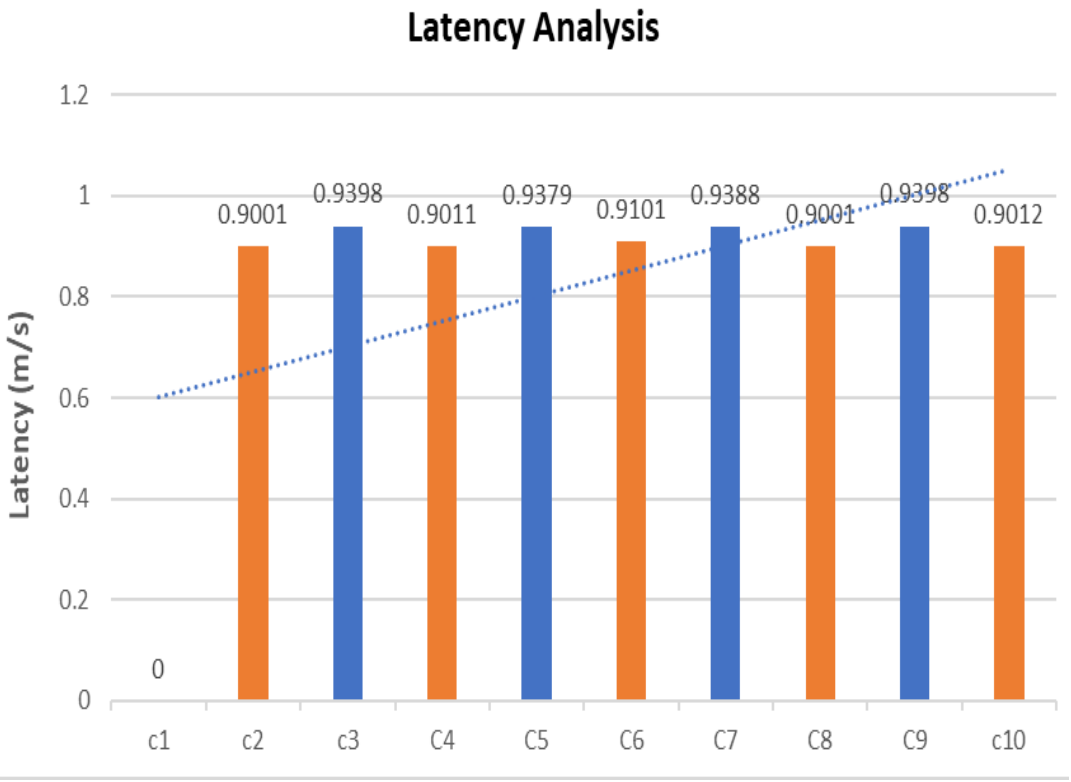


Figure 6.11: Controller latency analysis

In terms of the performance measure, we can confirm that the proposed SDN-FTF had the best performance in terms of using Ryu controllers during fault tolerance. From the experiments, it is evident that the proposed SDN-FTF shows better performance when using Ryu's controllers than OpenFlow. This is the same phenomenon that was also discovered during the evaluation of the total latency analysis for the SDN-FTF in both topologies shown in Chapter 5. Figure 6.12 shows the network overview after using the *dump* command in the Mininet command line to see all the nodes and their IP address in the simulated network using Ryu's controllers.

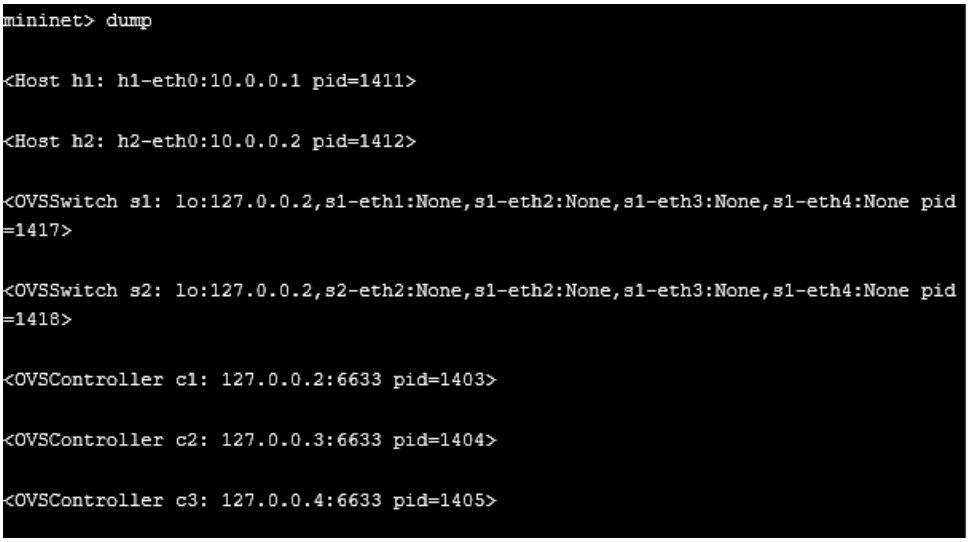


Figure 6.12: Network overview after the election of the new primary

From the results in Chapter 5 and Chapter 6, it is evident that when using placement and an expanded number of controllers exhibits a low latency rate in the network performance using Ryu's controller. This suggests that the system performs better than the OpenFlow controllers based on the performance analysis in terms of latency and throughput. Low latency is critical for real-time applications and network services as it directly impacts the responsiveness and overall performance of the SDN-FTM. In addition, it demonstrates that the time taken by the controller in the proposed SDN-FTM to process incoming controller messages is low and this reduces time delay in the event of a failure. In addition, the amount of data that can be transmitted through the SDN-FTM per unit of time is faster in handling network traffic efficiently.

Understanding the interpretation and the implication of the proposed SDN-FTM controller's throughput and latency analysis is crucial for determining the capabilities and constraints of the proposed SDN-FTM in a network environment. The results of these studies have a broader implication and have the potential to have a big impact on the proposed SDN-FTM's performance, dependability, and overall user experience. To verify that the proposed system can manage a large volume of real-time traffic, throughput, and latency studies are essential. Throughput analysis assures that the controller can react rapidly in the case of a network failure or fault. Scalability, fault tolerance, throughput, and latency are crucial for the proposed SDN-FTM to function optimally in terms of user satisfaction. By comprehending these results, network administrators may make informed decisions, optimise the controller's design, and increase the overall dependability and efficacy of their system.

6.7 Comparison of the proposed SDN-FTM with other systems

The permanence of the proposed SDN-FTM was compared with various frameworks based on the controller and architectural design. Among others, BFT-Smart [99], Ravana framework in [36], Fault Consistent SDN [30], SDN-SDNWS in [25], Rama [107], DSCA [9], FFDC [9], BOND [9] and FTlink [30], in Table 4.3. The proposed SDN-FTM introduced various SAs incorporated in four controllers (Chapter 5) that produced packet-in messages using network statistics while being monitored by SA and increased ten controllers with placement (Chapter 6) to test the throughput and latency performances of the controllers using Mininet. The results of the experiments show that the proposed SND-FTM has a higher throughput rate and lower latency when using multiple controllers in a distributed fashion. When compared to various works in the literature such as in [107], DSCA [25], FFDC [9], BOND [107], and [25], it is discovered that the throughput performance of the proposed SDN-FTM achieves twice higher performance with Ryu controllers than OpenFlow controllers. The performance of the proposed SDN-FTM improves compared to the previous framework presented within the framework of this study in the literature.

Table 6.8: Comparison of the proposed SDN-FTM with other systems with respect to latency and throughput

Type of Framework	**Focus Layer**	**Type of Controller**		**Type of Architecture**		**Problem/challenge**	**Solutions**	**Comparison with Respect to Latency**		**Comparison with respect to Throughput**	
		Single	Multiple	Centralised	Distributed			High	Low	High	Low
Rama [107]	Control Plane		X	X		Availability and scalability issues	Protocol for managing control messages	X			X
Fault-tolerant and Consistent SDN Controller [108]	Control Plane		X		X	Selection of the master and slave controllers	Database sharing between Master-Slave Controllers	X			X
DSCA [107]	Control Plane		X		X	Selection of the master and slave controllers	Architecture with Master-Slave	X			X
FFDC [9]	Data Plane	X		X		Single controller	Monitoring traffic to spot problems	X			X
SDN-FTM	Control Plane		X		X	Network stability and optimising resource	Monitoring and detecting faults using SA and recovering from faults using network statistics		X	X	

6.8 Summary

This section presented the controller placement implementation and simulation. The goal was to achieve the best placement that optimises the number of controllers needed to be deployed in the proposed SDN-FTM as well as determine the optimal locations to place them in the network. The focus in doing so is to reduce the average latency of the controllers, increase reliability, and ensure high availability in the network. To obtain this, we employ the *city-block* distance metric in MATLAB using K-means and K-medoid clustering to solve the question of how many controllers must be deployed and where they were placed, where they need to be placed in the network. This solution is based on a medium and large-scale network that requires a reasonable number of controllers to be deployed for the optimisation of the network performance and to always ensure of availability of resources. After evaluating the placement, the proposed SDN-FTM was tested using simulations in terms of its performance using Ryu and OpenFlow controllers. The results show that Ryu's controllers' performance was the best compared to OpenFlow. Lastly, a comparison analysis was done for experiences in Chapters 5 and 6 as well as a theoretical comparison of the proposed SDN-FTM with other similar systems in the literature.

Chapter 7

SDN-based On-Demand Security Services Framework

7.1 Introduction

The needs of today's information and communication technology (ICT) service providers and end-users cannot be met by traditional network architecture. Information risk management and security continue to pose substantial challenges for network service providers, corporations, organisations, and end-users within traditional network infrastructures. To expedite the deployment of their applications and network resources, an increasing number of businesses are turning to various cloud delivery options, including Infrastructure as a Service (IaaS), Hardware as a Service (HaaS), and Software as a Service (SaaS). To safeguard tenant applications and network resources, cloud providers rely on network security middleboxes such as firewalls (FW), intrusion detection and prevention systems, and anti-virus gateways [23] [34]. In cloud data centres, it is crucial to counteract threats to network security within multi-tenant infrastructures. However, the existing inflexible control mechanisms of security middleboxes, as highlighted in [10], impede flexible orchestration. This constraint limits the ability to provide flexible and on-demand protection in virtualised data centres [10] [34]. SDN presents a more streamlined approach to coordinating security middleboxes using OpenFlow [109]. Users can directly program, coordinate, control, and manage network resources using various SDN-based techniques. This empowers the SDN controller, a specific network component, with control over network resource management.

With the advent of SDN and Network Function Virtualisation (NFV), network operators now have greater flexibility in operating and designing their networks, overcoming legacy issues with conventional networks. The SDN controller centrally manages control over network resources in a logically centralised manner through interfaces designed for network resources. It also manages and configures distributed network resources, offering SDN applications an abstract view of the network resources. Through these interfaces, SDN applications can be configured to automate operations, including the management of abstract network resources, security policies, and services [110] [34]. Both SDN and NFV technologies are revolutionising the development of today's IT infrastructure, not merely by transforming network infrastructure from complex physical entities to virtual and programmable components [110] [34].

SDN and NFV enable a software-driven service that guides applications correctly and provides an abstracted sequence of service functions tailored to various service requirements. In this chapter, we introduce the concept of SDN-Based security services, underpinned by NFV and SDN technologies. Leveraging NFV technology, the proposed architecture establishes an on-demand security service platform. This platform empowers IT service providers to deliver dynamic, adaptable security to users and tenants of the SDN network at a reasonable cost [34]. The advantage of the proposed services is that users, including companies and organisations, can manage and select the essential security services required for their IT infrastructure. These services are provided by a service provider on an on-demand, cost-effective, and flexible basis. This solution addresses the constraints of traditional legacy network administration. The motivation for this concept stems from the challenges users (companies and organisations) face in managing network requirements and selecting appropriate security services for their IT infrastructure, with the aim of minimising costs and simplifying management [34]. In the event of incidents, needs, and attacks, users have the option to request additional services and utilise SDN programmability to dynamically modify their network policies.

7.1.1 Outline

This chapter presents the design for the proposed architecture for SDN-based security services based on SDN and NFV technologies. It outlines the specifications of the system and problem formulation for the design. In addition, it gives the objectives, and the requirements of the design and gives some directions for future research [34].

7.2 Overview of the proposed on-demand security services

To address security concerns and optimize ICT infrastructure costs for enterprises and organizations, this section presents a proposed architecture for SDN-based security services, utilizing NFV technologies. This architecture delivers comprehensive, on-demand security services that are customized to user needs. These services include the precise translation of network policies into flow entries, effective enforcement of mandatory network policies, support for packet data scanning detection, and other specialized security services. To enhance cost-efficiency while preserving network speed and security, the architecture provides standard security services such as authentication, authorization, confidentiality, integrity, software patching, and antivirus solutions. These services are all customizable based on user requirements [34].

The proposed on-demand security services, rooted in SDN, consist of several key components: the user system, security controller, SDN-security service provider's management system, NFV infrastructure, and a suite of Network Security Functions (NSFs) [34]. This architecture integrates NFV principles with SDN technology, leveraging the programmable nature of SDN to define responsibilities between the NFV controller and SDN controllers for enforcing security requirements. NFVs enforce security rules associated with NSFs based on their security capabilities, while SDNs enforce packet filtering rules that can be translated into packet-forwarding directives [34].

Therefore, the design makes the following contributions:

1) It proposes an architecture for on-demand security services that features a set of standard interfaces. This allows users to deliver security services on a Software-Defined Networking (SDN) platform swiftly and effortlessly [34].
2) It enhances on-demand service delivery on the SDN network platform. The design transitions from manual security enforcement to a security model with a high degree of service customisation and adaptability, including on-demand security services. Additionally, it introduces network slicing using both SDN and Network Function Virtualization (NFV) technologies [34].
3) The proposed design constructs the on-demand service using network slicing as an optimisation strategy. This offers users, including businesses and organisations, the ability to accommodate diverse on-demand service requirements through flexible service chaining and provisioning. In doing so, the design utilises the resource orchestration capabilities of SDN network services [34].

7.2.1 Motivation

Traditional network services are costly, challenging, and time-consuming to manage, typically requiring manual processes that are prone to error and can limit your organisation's productivity. Many businesses have already invested a great deal of time and money into developing flexible computing and storage environments, but they have not done the same for their network settings [34]. The network serves as the backbone of the entire IT system; hence this strategy is incorrect. Without the network, automating storage and computing environments causes an agility bottleneck and lowers IT infrastructure performance [34]. The risks associated with inadequate digital asset protection are a significant concern for businesses and organisations, and managing these risks can be costly [34]. To address these challenges, organisations can implement an SDN use case to alleviate security concerns, improve network

management, and realise cost savings. By utilising SDN and NFV technologies, businesses can leverage the benefits of both to achieve automation, scalability, and flexibility in security services, while maintaining customisation and control over on-premises operations [34]. This approach, which capitalises on SDN and NFV, is built atop network slicing. This allows businesses to simply specify their security needs, subscribe to the services they use, and deactivate others, rather than developing new services. The motivation for this concept stems from the challenges businesses and organisations face in managing network requirements, selecting the appropriate security services for their IT infrastructure, and doing so at a reasonable cost and with minimal management stress using network slicing [110] [34].

Therefore, this chapter introduces an architecture for SDN-based on-demand security services that integrates NFV technologies and SDN. This architecture is designed to assist companies and organisations in overcoming challenges related to the management and selection of essential security services for their IT infrastructure. The proposed solution prioritises cost-effectiveness and flexibility, catering to the needs of SDN service providers. To meet user performance expectations, the proposed design enhances flexibility in terms of scaling security services resources up or down and reconfigurability. It does this by introducing on-demand services that leverage network slicing for improved security control programmability and dynamic enforcement of security policies [34].

7.2.2 Design objectives

The goal of the proposed design is to make use of users' on-demand security services cost-effectiveness while satisfying their specification requirements and security demands [34].

The objectives for proposed SDN-based on-demand security services are as follows:

i. It should allow users to have subscription options for the security services that they need to help them defend their IT infrastructures from suspicious and malicious network attacks [34].
ii. It offered users (businesses and organisations) the option to improve flexibility in terms of scaling up or down security service resources [34].
iii. It should allow users to be able to use network monitoring and load balancing tools that calculate the cost of resources used for security services and automate security management to make dynamic resource selections that consider load balancing for the best network performance [34].

iv. It provided users with configurability in terms of security control programmability and dynamic enforcement of security policies. This allows users to protect their IT infrastructure independently, without the need for a network administrator's involvement [34].

7.2.3 Security services

Because of the invisible boundaries that are now created between shared logical and virtual entities across many internet users, traditional security measures and endpoint protection for information and network resources are no longer sufficient. This section presents and discusses various security services that can be demanded and provided by the proposed ODSS [34]. The focus of the design architecture is to address users' requests for security problems relating to data, resources, applications, and clients' physical and virtual network infrastructures. The proposed ODSS offers security services that address the five major domains of security concerns such as availability, confidentiality, authentication, data integrity, and non-repudiation [4] [34]. These security services are offered under the service provider management system using the security services module and security service management. In the security service module, users can subscribe to a basic service that needs day-to-day operation of the network, and in the security, management is services that are needed for network management and other security enforcement for their systems. These security services address specific threats and attacks that organisations and businesses face in their operation. The following section discusses all of them in detail and Figure 7.1 provides an illustration of security services that can be demanded and provided, together with the threats and that they are addressed [34].

i. Availability

The most crucial component of ODSS is the availability of services for users (businesses and organisations) because it directly affects all safety applications. The primary role of availability is functionality management, and its security must ensure that the network and other applications continue to function in the event of an error or harmful conditions for the end users. Information and infrastructure (physical or virtual) accessibility is a crucial need for all business and organisation systems because a lack of availability compromises the effectiveness of the end-users [4] [34].

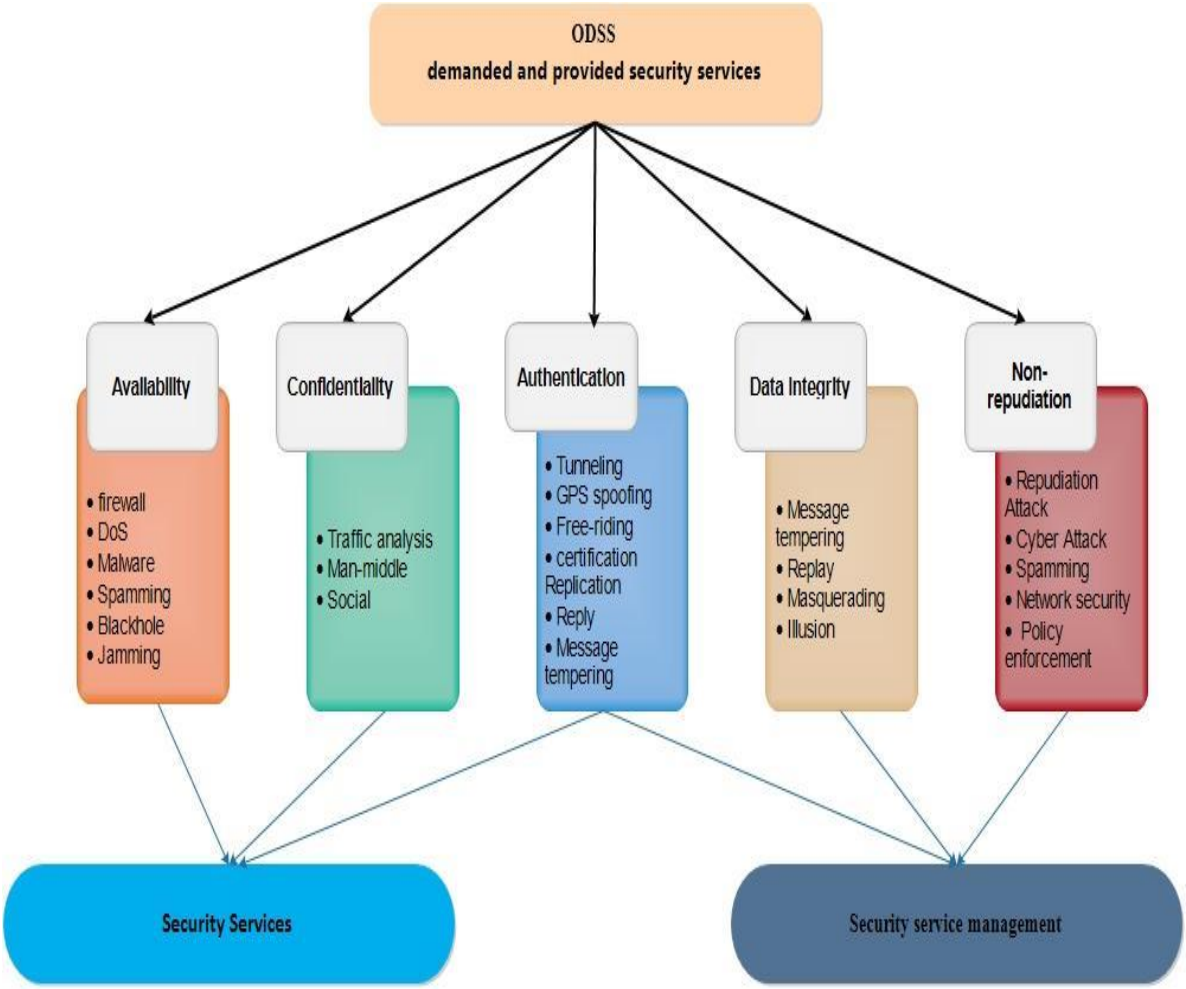


Figure 7.1: ODSS security service available on demand [34]

ii. Confidentiality

Based on certificates and shared public keys, secrecy for businesses and organisations has to ensure that only the specified receiver has access to the data, preventing outside nodes from doing so until the designated user has received the sensitive data. Confidentiality ensures that all exchanged messages can be encrypted so that only authorised users can decrypt them by utilising certificates and sharing public keys [4] [34].

iii. Authentication

Authentication is crucial because it stops business information and infrastructure from alleged attacks in the network. For businesses and organisations to defend against external users infiltrating the system, authentication is needed [4] [34].

iv. Data integrity

It ensures that the message's content is not alerted while being transmitted. Data integrity is supported by the usage of public key infrastructure and cryptography procedures. Data integrity makes sure that the data transmitted is accurate [4] [34].

v. Non-repudiation

This ensured that, in the event of a disagreement with the message's sender, the recipient did not decline to participate in message transmission and reception [4] [34].

7.3 On-demand security services architecture

The following section presents the system requirements of the proposed OBSS, design architecture, security services deployment processes, and users' security service creation as well as a theoretical comparison of various framework evaluations in the literature [34].

7.3.1 System requirements

The enforcement of security service regulations for on-demand using SDN and NFV technologies is divided into the proposed SDN-based security service presented in this chapter. Simple packet filtering rules that can be converted into packet forwarding rules are specifically enforced by SDN security controllers. Although NFV demands that NSFs have security capabilities that the user accesses through network slicing, it also enforces NSF-related security standards [34]. The requirements for the proposed SDN-based security service on-demand are outlined below to implement its capability:

- To effectively implement users' on-demand security services, the proposed SDN-based security service must enable the permeability of network resources. This ensures that network resources can be accessed and utilized efficiently, enhancing the overall effectiveness of the security services.
- The proposed SDN-based security service on-demand must support the orchestration of SDN applications that safeguard users' infrastructure and security service on-demand requests.
- The proposed SDN-based security service on-demand must use network slicing to give users access control and autonomous management of their security services on-demand monitoring.
- To protect users' IT infrastructures, the proposed SDN-based security service on-demand must give users logically centralised control over security services.

7.3.2 Architecture

Our proposed solution's architecture is designed to be self-contained, virtual, and logical in nature. To cater to the needs of organizations and businesses for security services on demand, we incorporated a network slicing layer over the infrastructure of SDN and NFV. This strategy utilizes abstraction and mechanisms of security service. Network slicing, a method frequently employed in 5G systems, allows network operators to dynamically manage network resources in accordance with the services offered to subscribers, other users, and third-party customers [4][34]. The architecture of the on-demand security service based on SDN comprises several crucial components: the system user, the management system of the service provider, the security controller, the NFV forwarder, and the network slice layer, as illustrated in Figure 7.2.

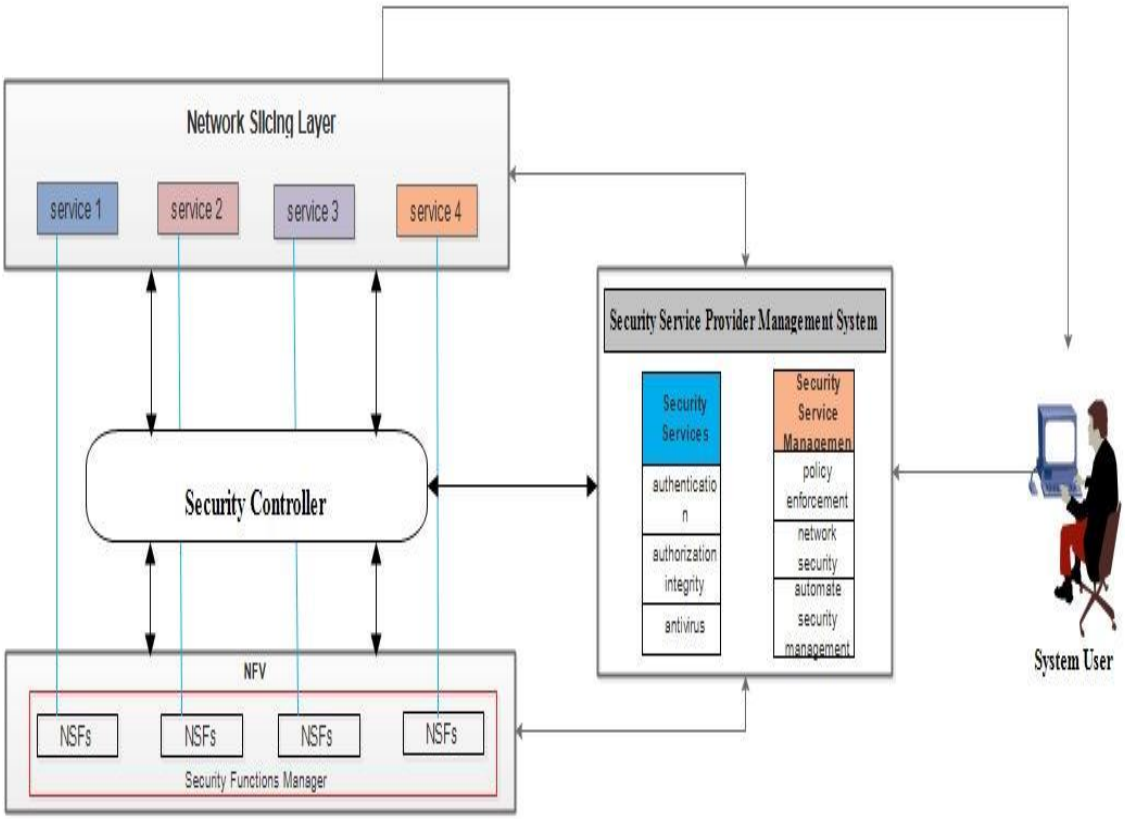


Figure 7.2: Proposed on-demand security services architecture [34]

The idea of the above proposed ODSS was motivated by the work of [109]. The proposed framework in [109] offers security services based on network virtualisation approaches using SDN and interface network security functions (I2NSF). However, it emphasises DDoS attack mitigation and centralised firewall systems. The purpose of this framework was to protect

network resources by controlling suspicious and dangerous network traffic based on the existing systems [109]. In [109], cloud-based security services were presented utilising an interface for network security features. By incorporating SDN, the author created and developed an I2NSF architecture to increase system efficiency. However, in this proposed design, the I2NSF makes use of a set of standard interface security applications using heterogeneous NSFs [109]. A secure SDN framework for IoT was also proposed in [109] to solve the limitation of network legacy by designing a secure architecture for IoT in an SDN-based network. This design employs a blend of legacy interfaces, a programmable layer, and an SDN controller. Ad-Hoc clients utilize associated nodes via their embedded SDN-compatible controller to secure their network infrastructures. The proposed ODSS adopted a few components [109] as well as added the concept of network slices used in [111]. The idea is to design a flexible system, programmable and dynamic to respond to businesses' and organisations' needs following their security policy enforcement and resources.

The subsequent section elaborates on the various components of the proposed On-Demand Security Services (ODSS) architecture depicted in Figure 7.2, along with their functionalities:

- **System User (SU):** In this context, the user refers to the network administrator of the organization or company, whose responsibilities include enforcing and requesting security services on demand for the entity. The user sends a request to the service provider, outlining the security service required to safeguard their IT infrastructures. This request encompasses a variety of security services and a high-level security policy [34].

- **Service Provider Management System (SPMS):** This function is performed by a third-party provider who specializes in providing security services in an on-demand manner. In accordance with the requirements of the users, this entity oversees the dynamic lifecycle of NSF instances. The SPMS establishes user accounts and sets up their on-demand security services using NSFs, which are automated through the programmable capabilities of SDN, and assigns slicing to the user. This allows users to take advantage of a variety of network slicing options offered by the service provider, customized to meet their on-demand security needs [34].

C. The subsequent section delineates several activities that the SPMS must undertake to deliver on-demand security services to subscribed users [34]:

- ***User accounts:*** This is an identity created for a business or organization to gain system access. In this scenario, it signifies an account utilized by the business system administrator to monitor and manage the system on behalf of the subscribed business or organization [34].
- ***On-demand security services:*** These encompass a variety of service offerings determined by the user upon subscription, such as firewall, anti-virus, mail filter, DDoS mitigation, and web filter [34].
- ***SDN controller:*** This element provides the programmable capability to supply available on-demand security services to users, automating various user requests. It acts as the main component of the SPMS within the vendor's domain, managed by the service provider [34].
- ***Slicing allocation:*** This procedure involves assigning services to end-users or organizations by defining network requirements once the physical or virtual network resources are deployed [34].

D. **NFV Forwarder:** This component serves as the manager of the security function, employing numerous NSFs to conduct network security inspections. This is accomplished by the security controller enforcing policy rules and fulfilling requests for on-demand security services. The security controller and the network slicing layer can coexist with the NFV forwarder, a critical element of the design. The NFV forwarder, utilizing a security function manager, determines the role and sequence of each NSF, thereby providing flexible security services that are based on demand through network slicing [34].

E. **Network Slice Layer:** Built upon the SDN infrastructure, the network slice layer is conceptualized as a self-contained, isolated, and virtual network. It is composed of a series of NFV instances and resources that are interconnected to form a service function chain [34]. The proposed model includes multiple slices to offer users access to a variety of on-demand security services they have subscribed to, ensuring there is no interference with other users. The SDN controller facilitates flexible resource allocation and policy-driven network management, while also supporting instantiation on a per-user/per-slice basis [34]. Even though they share the same underlying physical network, each slice manages packet

forwarding independently based on the allocation by the service provider, without impacting packets from other slices [34].

F. **Security Controller:** This core component operates in a manner akin to network operator management in conventional networks. The security controller orchestrates and oversees the NFV and network slicing layer, collaborating to enforce on-demand security services set up by the service provider for the user [34]. Upon receipt of a request for policy enforcement or any other on-demand security service from a registered user, the security controller identifies the types of NSFs needed to implement the on-demand services [34]. It subsequently formulates security rules for each required NSF within a specific network slice and updates each NSF's configuration with the newly created policy rules. Furthermore, the security controller monitors details such as workload status and network access, while supervising the NFV and network slicing layer. For the algorithms involved in this process, please refer to Figure 7.3 [34].

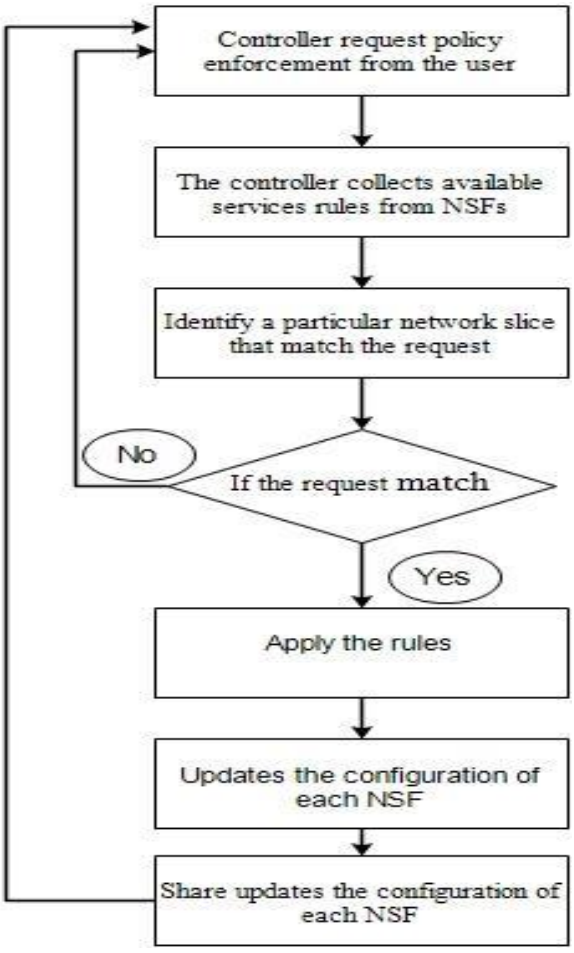


Figure 7.3: Proposed ODSS security process [34]

7.3.3 Security services deployment

The deployment of the proposed security architecture is illustrated in Figure 7.4. This architecture can be employed to offer businesses and organizations on-demand security services, leveraging SDN and NFV within a cloud environment [34].

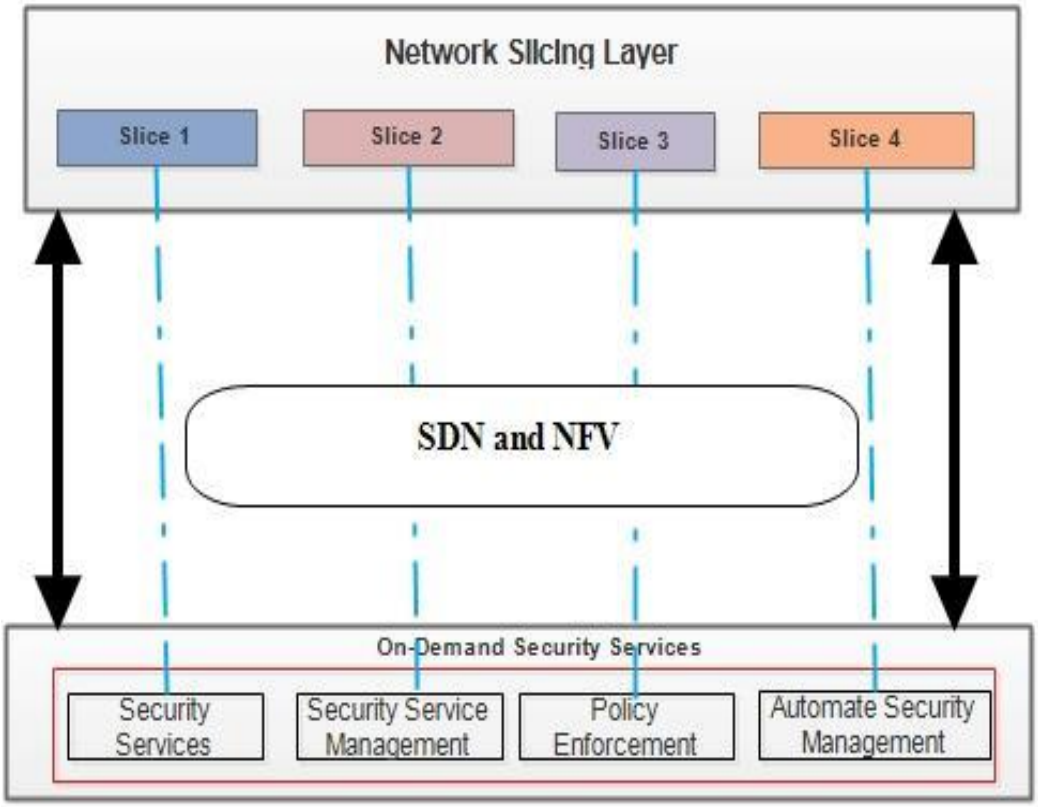


Figure 7.4: On-demand security services deployment [34]

The network slicing layer is composed of numerous slices (for instance, slice 1, slice 2, and so on), each offering distinct services such as security services, management of security services, policy enforcement, and automated security management, as illustrated in Figure 7.4 [34]. The assortment of slices within the NVF is represented as:

$$S = [1, 2, 3 \ldots S] \quad \text{(Equ.7.1) [34]}$$

In this system, each slice 's' contains a set of system users 'SUs', represented by the symbol [34]. This can be expressed as:

$$Us = [1, 2, \ldots Us] \quad \text{(Equ. 7.2) [34]}$$

For on-demand security services, each slice ‘s’ submits a request to the Service Provider Management System (SPMS). In our model, ***Rs min*** and ***Rs max*** represent the minimum and maximum services related to the slice, respectively [34].
A priority $\mathbf{P}^{s}$ is defined with the restriction that:

$$\sum s \in S\, Ps = 1 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \text{(Equ. 7.3) [34]}$$

This defines the characteristics of each slice. Similarly, each user ***U*** of the slice **S** denoted as ***US***, is identified by a priority μus, where $\sum us \in Us\ \mu us = 1$ …… (Equ. 7.4) [34]

In the suggested model, the SDN controller functions as the system’s provider of security services, tasked with the deployment of all services on demand. It oversees the logical association between users (such as organizations and businesses) and the distribution of requested security services. For example, it assigns services to various users, enabling the provision of services to multiple users [34]. Furthermore, it manages and coordinates users’ access to shared infrastructures. In this scenario, a user is a logical entity charged with owning and administering one or more security services. Based on the user’s preferences, the NFV Forwarder can enrol users in services operating in the SDN across one or more slices on the network slice layer [34]. The Security Function Manager assesses ongoing communications based on the most recent packet property information supplied by the classifier [34]. If any problems are detected, it dynamically deploys the service in accordance with the decision policy, as per the on-demand request. Additionally, the Security Function Manager can distribute NFV resources across the substrate network, creating multiple network slices (such as slice 1, slice 2, etc.) to provide security services to a variety of users [34]. Each user (for instance, organizations or enterprises) may possess and manage a network slice, such as slice 1 or slice 2, depending on the subscribed security service. Within the proposed framework, users (organizations and companies) have the flexibility to deploy their network services or permit external users or service providers to instantiate their services atop the NSFs [34]. Also, the Security Function Manager identifies the quantity and types of NSFs needed by users. If the current deployment is deficient in the correct number, NFV is alerted to commence a new deployment. Upon receipt of the notification from the Security Function Manager, NFV processes the requests and assigns security services to the users. The Security Function Manager continuously monitors the status of network slice resources and modifies the current on-demand security service deployment based on NFV status. Moreover, NSFs can be stacked

on top of each other. For instance, the NSF of a company using slice 1 can be constructed over the NSF of an organization utilizing slice 3, and vice versa [34].

7.3.4 Users' security service creation

Via the NFV, the security service provider empowers users to request and make use of their user accounts. Each user is allocated a unique access ID, which can be created and linked with the corresponding ID of the on-demand services. Subsequently, user administrators can craft their requests for the on-demand security services necessary for their IT infrastructure. The procedure for establishing user accounts encompasses the steps depicted in Figure 7.5 [34].

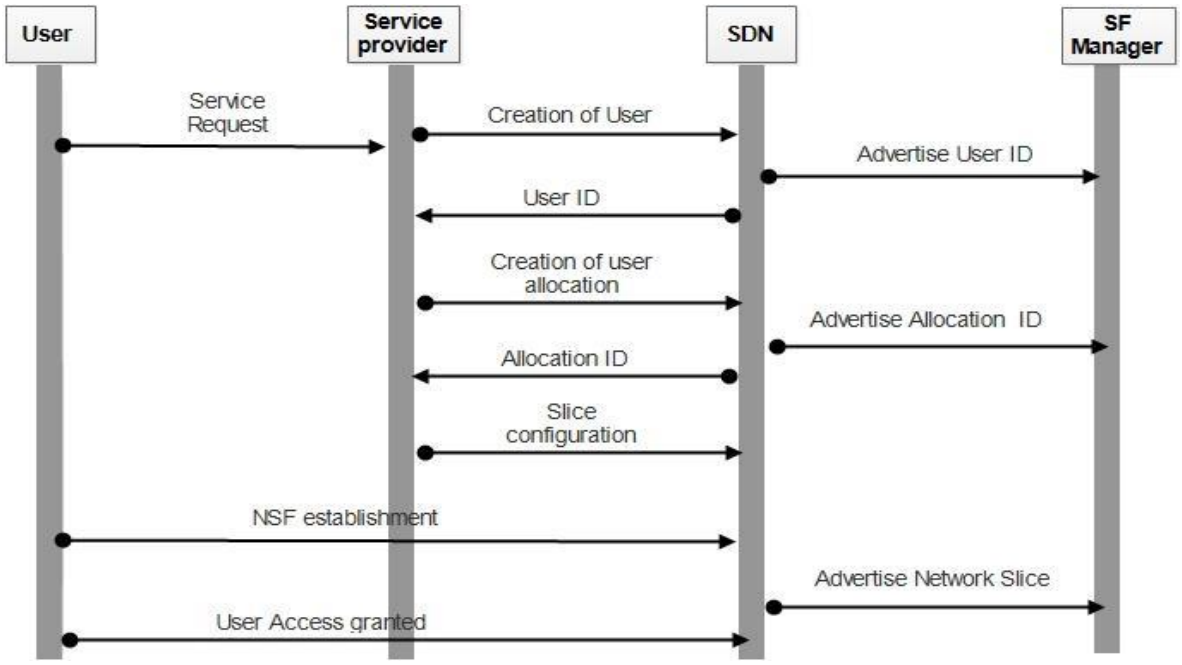


Figure 7.5: User service request creation [34]

As depicted in Figure 7.5:

i. The user submits a request for the security service to the provider.
ii. The operator of the security service provider generates the users' accounts.
iii. Service allocation is identified using the created user ID.
iv. The ID is sent to the user.
v. The role and sequence of each NSF are defined to deliver on-demand security services to users.

vi. A dedicated network slice is assigned for the requested security services.
vii. The network slice ID is sent to the user.
viii. Access is granted to the user.

7.4 Framework theoretical evaluation

In traditional networks, all traffic flows typically follow the same path, irrespective of whether the packet requirements are identical or not. However, with the use of SDN and NFV technologies, network traffic flows can be directed to their desired services in response to customer requests. This approach also serves as a source of inspiration for various research focuses. Numerous researchers have made contributions to the literature, proposing on-demand security service architectures based on a variety of technologies, including SDN, NFV, cloud data centres, NSFs, and I2NSF [34]. This section provides a comprehensive analysis and evaluation of selected existing architectures or frameworks, including the proposed ODSS[34]. To offer ICT service providers a dynamic and adaptive secure service on the SDN network as well as the provision of a security service function chain, [112] proposed an architecture for on-demand security services within a software-defined network, leveraging virtual network capabilities [34]. To ensure that the protective advantages of network security are realised, the system concentrates on traffic flow pathways, allowing flow from the source to travel through a series of personalised information security services before accessing the destination. To further realise information security services with alternative deployments and differentiation, the system offered flexible customisation on every link [34] [112]. With more freedom in this solution, the SFC operates similarly to a mobile network value-added function. Self-defined, value-added service may be provided by this architecture [34] [112]. In [113] Tualatin was presented as a streamlined framework for providing security services in multi-tenant data centres [34]. By jointly designing the hardware and software, it satisfies the security requirements of many scenarios. The design offers fine-grained security protection in dynamically changing network topologies where switches and security middleboxes are programmatically managed by logically centralised controllers by utilising SDN and OpenFlow approaches. The Tualatin framework could easily be linked with various cloud management systems using its service-level APIs [34] [113].

A framework for 5G verticals, utilizing network slicing techniques and offering virtual security as a service, was proposed in [113]. This framework explores the potential of both SDN and NFV to provide on-demand protection for a network slice. It addresses the additional security challenges introduced by the flexibility and elasticity of these technologies [34]. The primary

objective of this framework is to provide optimal resource allocation that effectively manages the security strategy of the slice and assesses the security overhead in virtualized environments [113] [34]. Based on their findings, the authors recommended an auto-scaling solution. This solution considers each Virtual Network Function's (VNF) unique performance needs, traffic load forecasts, and instance start-up time to initiate scaling operations [113] [34].

To tackle the issue of on-demand service function chaining in IPv6, the authors proposed using Segment Routing over IPv6 (SRv6) [34]. This framework allows SRv6 to guide the Service Function Chain (SFC) path for detecting malicious attacks and controlling traffic. This approach effectively enhances IPv6 network security and service quality, as demonstrated by experimental data [34]. It enables network service managers to dynamically divert traffic according to their plans, thereby increasing the efficiency of network service access. It also addresses the problem of all traffic having to pass through all information security nodes [34]. Additionally, the authors proposed using a traffic controller VNF to enhance the network security and traffic engineering management mechanism [34].

In [10], the authors proposed the design and development of an Interface to Network Security Functions (I2NSF) architecture. They suggested enhancing its efficiency by integrating SDN [34]. The primary goal of the I2NSF architecture is to provide standardized interfaces for managing and controlling the NSFs of various suppliers [10] [34]. Furthermore, they developed general, industry-standard data and information models with vendor-neutral syntax rules and data structures. These models accurately represent the security policy rules for NSFs from different manufacturers with similar security capabilities [10] [34]. Similarly, the authors proposed an architecture for security services that utilizes SDN and I2NSF [10]. This architecture features a centralized firewall system and a DDoS attack mitigation system to protect clients' allocation of network resources by controlling suspicious and risky network traffic [34]. Compared to the ODSS proposed in this study, several of the aforementioned frameworks share similarities and overlap in their design components [10] [34]. However, the integration of network slicing techniques makes the proposed ODSS more unique and tailored to users' on-demand requests [34].

To show improvement in the proposed framework with others existing in the literature, table 7.1 gives a details comparison in terms of the technology used, key focus, platform, problem addressed, solutions and improvement in their contributions [34].

Table 7.1: Comparison of the proposed ODSS with improvements in existing models in the literature [34]

Type of Framework	Technology	Key focus	Platform	Problem	Solution	Improvement
Interface to Network Security Functions for Cloud-Based Security Services [10]	NSFs. And SDN	Optimisation of the Vendors' security service enforcement	Cloud-based security service environment	Enforcement of Security services for cloud-based vendors	I2NSF architecture to standardise the interfaces to NSFs for Vendors to simplify management of the NSFs	SDN-integrated I2NSF architecture and its security applications to improve its efficacity.
SDN-based Security Services Using Interface to Network Security Functions [114]	SDN and I2NSF	Centralised firewall system and DDoS-attack mitigation system	SDN network	Protection of network resources attacks	Controlling suspicious and dangerous network traffic and attacks	Security services based on network virtualisation
A security Service on-demand Architecture in SDN [112]	SDN and NFV	Control of traffic flow and delivery of an abstract sequenced set of functions according to various service requirements	SDN network	Protection against security attacks	Novel architecture as a security service platform with on-demand virtual network functions	Security service function chain to provide secure service on the SDN network
Tualatin: Towards Network Security Service Provision in cloud datacentres [112]	A virtual router and NFV	Virtual private cloud (VPC) infrastructure	Cloud-based	Security protection in multi-tenant infrastructures deployed in cloud data centres.	Delivery of security services in multi-tenant data centres	Provision of customised network security services
Virtual Security as a Service for 5G Verticals [95]	SDN and NFV	5G wireless network access systems	Multi-cloud environments	Service resource allocation, monitoring, and flow control mechanisms	Levering both SDN/NFV to deliver security as a service (SECaaS)	Provision of a secure network slicing on-demand
-Demand Service Function Chain Based on IPv6 Segment Routing [34].	IPv6 (SRv6) and Service Function Chain (SFC)	5G innovation	Cloud-based	Malicious attack detection and traffic detection	Use SRv6 technology to implement a Service Function Chain (SFC) combined with network security applications.	Integration of virtualised security service and SRv6 traffic mechanism for the application of SFC.
SDN-FTM	SDN, NFV and network slicing	Management of network resources such as Information and security risk	SDN network and cloud-based environment.	Protection of network infrastructures and information against Malicious attack	Subscription of users to create Security services on-demand based on SDN and NFV in the cloud environment.	Unification of the user system through network slicing, security controller, SDN-security service provider's management system, NFV, and an assortment of NSFs.

7.4 Summary

This chapter presents a proposed on-demand service based on SDN, employing an SDN controller and NVF. This proposed framework distinguishes itself in terms of design and application adaptability and uniqueness compared to existing designs in the literature. The architecture of the proposed ODSS, inclusive of all components and their functionalities, is elaborated [34]. Operating concurrently within the SDN network platform, the proposed architecture enables the delivery of on-demand security services in a cloud-based fashion [34]. It guarantees consumers, companies, and organizations access to on-demand security services, cost-effectiveness, deployment efficiency, and flexibility [34]. This chapter illustrates that the proposed architecture streamlines the process for consumers, companies, and organizations to secure necessary security services for their ICT infrastructure from a service provider, adaptable to their specific needs and demands [34]. To ensure its uniqueness and superiority over existing designs in the literature, a theoretical comparison was undertaken, detailing various features of the architecture. The subsequent step involves substantial refinement of the design to bolster the security of the infrastructure within the SDN platform.

Chapter 8

Summary, Conclusion and Future Work

1.1 Chapter outline

This chapter presents a broad summary, conclusion and recommendations for further research, and potential future projects. It provides a detailed summary of the steps taken to accomplish the study's goals. It also draws attention to the areas that need further study for subsequent work.

1.2 Research summary

In the context of SDN, this study presents various features that can address fault tolerance and its applications in enabling on-demand security service. To provide a comprehensive understanding of the topic, we also provide a fundamental background on fault tolerance and related techniques. This study aims to design and evaluate a controller fault tolerance model using simulation tools as a proof of concept. In addition to evaluating the design using placement, an on-demand security service framework based on SDN technologies is proposed. The research body around SDN architecture layers identifies three core areas of focus in organizing and categorizing the research: the data plane, control plane, and application plane. It was observed that each layer of SDN has its shortcomings, and the approach to addressing fault tolerance problems may vary between them. This study focuses on faults impacting the controller plane, with the implemented solution specifically designed to address this area of concern.

In the thesis, Chapter 1 presents the background information and arguments for the significance of this study. It defines and addresses the problem description, goals, and objectives. Equally, Chapter 2 reviews various works of literature that address issues related to fault tolerance in SDN, cloud computing, controller placement, and other aspects of fault-tolerant techniques. This chapter explains these techniques using results from literature studies, as well as comparisons of various key concepts of this study and content analysis. Similarly, Chapter 3 delves into the design science research used in this study and the various iterations and steps taken to design, develop, and evaluate the proposed SDN-based FTM and the proposed on-demand security services. Chapter 4 encompasses the empirical analysis utilized to select the optimal approach in designing the proposed system. This chapter employs a series of design

principles, components, and mechanisms for the development of the artefact. Additionally, the developed design was tested using simulation tools based on various parameters and deployed in different scenarios to evaluate fault tolerance using small-scale network scenarios in Chapter 5. Chapter 6 presents the formulation and evaluation of the placement problem using the proposed SDN-FTM. Furthermore, the placement was used to test the proposed design in medium and large-scale network scenarios. Lastly, Chapter 7 presents the design of the proposed on-demand security services framework as a way to address some challenges faced by organizations in securing their network infrastructures and information.

To ensure that the research objectives were met, this section demonstrates the various objectives of this research and how they were achieved in each chapter:

RO1: I***dentification of existing work in the literature on SDN, its challenges and applications, existing SDN fault tolerance techniques, and SDN-based security services frameworks.***

In Chapter 2, a comprehensive review of existing literature was conducted to address RO1. Various sources such as journal articles, conference proceedings, magazines, and books were examined, focusing on fault tolerance, the techniques and frameworks used for fault tolerance, and literature on on-demand security services. An overview of traditional networks was presented, contrasting them with the programmable capabilities of SDN. The focus was on static and dynamic network traffic, which is significantly impacted by today's high demands for internet usage. The deployment of SDN fault tolerance was covered across various fault domains, including hardware failures and links in the control plane, along with other techniques to address these issues. In addition, other factors affecting fault tolerance and SDN were reviewed in detail in this chapter. Various issues related to existing fault tolerance were discussed to establish a solid foundation for the study.

A comparison of various fault-tolerance frameworks was conducted to evaluate their design and performance against the proposed SDN-FTM. An extensive comparison was made in terms of the techniques used, the type of controllers, the layer of the SDN controller that the problem was focusing on, whether a single or multiple controllers' approach was deployed, and their limitations. This comparison aimed to provide a comprehensive understanding of the strengths and weaknesses of each approach, thereby informing the development of more effective fault tolerance mechanisms in SDN environments.

RO2: Propose and design an effective fault tolerance model for SDN that ensures controllers' reliability and availability.

RO2 was accomplished in Chapter 4, where various assumptions were presented to ensure that the system design is fault-tolerant. Different design principles leading to the selection of FTM were discussed, as well as the requirements of the proposed system. The architecture of the FTM was developed, and the functionality of each component was defined and explained in detail.

The operation of the SDN-FTM was elucidated at each level. The system was evaluated using all the design principles, and detailed descriptions of these principles were provided. This comprehensive approach ensured a thorough understanding of the system's design and operation, contributing to the achievement of RO2.

RO3: Configure the proposed FTM for small- and large-scale networks, implement and evaluate the performances.

RO3 was accomplished by implementing the system and evaluating the proposed SDN-FT model. The architecture design was tested in Mininet using various scenarios and customised parameters that align with the design principles, as detailed in Chapter 5. Various system properties were utilised during the simulation, along with evaluation metrics and other significant steps taken during the simulation. The evaluation was conducted in a small-scale network scenario, and the performance results were presented and discussed in detail. The results indicate that the design performance in the small network favours Ryu's controllers. The system was evaluated in terms of its throughput and latency performances to ensure that it addresses all the design principles.

Additionally, a controller placement strategy was formulated in Chapter 6 to evaluate the optimal position for controller deployment, addressing this objective. For the evaluation of the placement for the proposed SDN-FTM, an experimental setup was developed, and system properties were defined for the simulation of the placement. The placement evaluation was conducted for medium and large-scale networks using various evaluation metrics and simulation parameters. Comprehensive simulation steps were followed, and the results demonstrate that the selected controller position was optimal for the performance of the SDN-FTM.

Obj4: Proposed on-demand security services utilising SDN functionalities for adoption in an Internet or cloud.

This objective was accomplished by proposing an on-demand security service that enables internet users and organizations to subscribe to the security services they require to protect their IT infrastructures from suspicious and malicious network attacks. The on-demand security service was designed, and each system component was explained in detail, along with their functionality. Leveraging SDN and NFV technologies, these components were incorporated into the proposed SDN-based security service presented in this chapter. Furthermore, the requirements for the proposed on-demand SDN-based security service were outlined to demonstrate the implementation of its capabilities. This approach ensures a robust and flexible security solution that can adapt to the evolving threat landscape.

1.3 Research conclusion

The proposed SDN-FTM was designed following the design principles and validated using various components of the constructed system. The system was evaluated in small-scale network scenarios using the architecture presented in Chapter 4, with the modified backup controller having a stationary agent and using a distributed approach when deployed in medium to large-scale networks. Various parameters, techniques, and performance measures such as latency and throughput were used to evaluate the FTM in real-time using Mininet simulation tools. The focus was on achieving fault tolerance while minimising the average latency of the controllers and maximising the reliability with the high availability of the network. In addition, another goal was to accomplish the best placement that optimises the number of controllers needed to be deployed in the proposed SDN-FTM, as well as determine the optimal locations to place them in the network. To achieve this, the city-block distance metric in MATLAB was used by methods of K-means and K-medoids clustering. This solution was based on a medium and large-scale network that requires a reasonable number of controllers to be deployed for the optimisation of network performance and to always ensure the availability of resources. After evaluation, the results indicate that the proposed SDN-FTM system optimised network performance with OpenFlow controllers having a lower latency rate compared to Ryu's controllers. This minimum latency shows that the OpenFlow controllers' responses after failover are better in a small-scale network than in Ryu. However, in the medium to large-scale network, the evaluation reveals the proposed SDN-FTM achieved the best performance with Ryu's controllers compared to OpenFlow.

Furthermore, the proposed framework for SDN-based on-demand security services was developed using SDN and NFV technologies. This makes it more adaptable and unique compared to any previous design in the literature. This architecture operates in parallel within the SDN network platform, facilitating on-demand security services in a cloud-based or internet-provisioned manner. Its functionality assures consumers and organizations of on-demand security services, cost savings, and efficient hardware deployment, thereby offering flexibility. Essentially, by streamlining the system components, the process for consumers and companies to access the necessary security services for their ICT infrastructure from a service provider becomes simplified. This is adaptable to their needs and requests on demand. To strengthen the infrastructure within the SDN platform, this research suggests that this proposed design serves as a foundational model for the academic community. It can be used for implementing on-demand security services leveraging SDN functionalities.

1.4 Recommendation and future works

The research community has numerous opportunities to develop new fault-tolerance mechanisms, standards, monitoring, debugging, and testing tools to enforce fault-tolerance in dynamic networking environments capable of enduring carrier-grade reliability. Despite the infancy of SDN fault tolerance, drawing from existing literature on SDN-based security services, the goal of the proposed design and its architecture necessitates significant research efforts. This effort should focus on articulating strategy implementation, developing concepts of high-level system architecture, and exploring the development of the actual proposed system to bridge the gap between the current research progress in SDN-based security services.

From the results of the experiments on the proposed SDN-FTF and the evaluation of the proposed ODSS, the researcher proposed the following:

- The proposed SDN-FTM can be evaluated in real-time in a network with only Ryu's controllers and OpenFlow controllers, comparing their performances in terms of fault tolerance.
- Further research could be conducted to evaluate the performance of a controller having a stationary agent and a database for storing information, to compare its performance against the proposed study.
- The creation of a proof-of-concept implementation of the proposed architecture using a variety of open-source programs and different on-demand security service scenarios.
- Configuring the system components' SDN networks using various simulation tools.

- The proposed systems offer additional security that can be added to other Network Security Functions (NSFs).
- The creation and implementation of a web-based application that provides users with a speedy and easy way to request, configure, and monitor high-level security services.
- A closer examination of the possibility of creating a multi-user on-demand service.

The future development of a centralized SDN controller capable of monitoring and implementing real-time intrusion detection in high-speed networks presents a challenging task. Most SDN-based Network Intrusion Detection System (NIDS) architectures were developed primarily to detect malicious behaviour on Small Office/Home Office networks. It is crucial to emphasize that none of the techniques for implementing SDN-based NIDS covers critical intrastate or high-speed network infrastructure. We believe that researchers can implement Machine Learning/Deep Learning-based NIDS on critical infrastructure with increased SDN accuracy and scalability. We think that this comprehensive study could aid in the understanding of NIDS development within the context of SDN by Research and Development personnel.

9. Appendices

9.1 Customised SDN-FT Topology Python Mininet


```
{
   "application": {
      "dpctl": "",
      "ipBase": "10.0.0.0/8",
      "netflow": {
         "nflowAddId": "0",
         "nflowTarget": "",
         "nflowTimeout": "600"
      },
      "openFlowVersions": {
         "ovsOf10": "1",
         "ovsOf11": "0",
         "ovsOf12": "0",
         "ovsOf13": "0"
      },
      "sflow": {
         "sflowHeader": "128",
         "sflowPolling": "30",
         "sflowSampling": "400",
         "sflowTarget": ""
      },
      "startCLI": "0",
      "switchType": "ovs",
      "terminalType": "xterm"
   }
```


9.2 OpenFlow SDN-FT Controllers Setup in Mininet


```
controllers:
      {
         "opts": {
            "controllerProtocol": "TCP",
            "controllerType": "ref",
            "hostname": "c1",
            "remoteIP": "127.0.0.1",
            "remotePort": 6633,
  "ControllerStatus":”UP”,
         },
         "x": "355.0",
         "y": "173.0",
      }
```

9.3 Nodes Setup in Mininet


```
"hosts": [
        {
           "number": "1",
           "opts": {
              "hostname": "h1",
              "IP": "10.0.0.1",
              "nodeNum": 1,
              "sched": "host"
           },
           "x": "687.0",
           "y": "481.0"
        },
        {
           "number": "1",
           "opts": {
              "hostname": "h2",
              "IP": "10.0.0.2",
              "nodeNum": 2,
              "sched": "host"
           },
           "x": "227.0",
           "y": "485.0"
        }
    ],
    "links": [],
    "switches": [
        {
           "number": "1",
           "opts": {
              "controllers": [],
              "hostname": "s1",
              "nodeNum": 1,
              "switchType": "legacySwitch"
  "switchStatus":”UP”,

           },
           "x": "347.0",
           "y": "331.0"
        }
```

9.4 SA Sender Deployment

```
import socket import sys
Try:
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print ("HELLO message successfully sent")
        except socket.error as err:
        print ("HELLO message failed with error %s" %(err))
                Port = 6633
Try:
cntr_ip = socket.getcntrbyid("secondary controller")
        except  socket. gaierror:
        Print ("there was an error resolving the secondary controller")
sys.exit()
s.connect((cntr_ip, port))
print ("the controller has successfully connected to the primary")
```

9.5 SA Receiver Deployment

```
import socket
        s = socket.socket()
print ("HELLO message successfully sent")
        Port = 6635
        s.bind((",port))
print ("SA binded to %s" %(port))
        s.listen(3)
print ("SA is listening")
        While True:
                C, addr = s.accept()
Print ('received a connection from' addr)
        c.send('message received, and communication granted' ,encode())
s.close()
break
```

9.6 SA Action Method

```
import socket
s = socket.socket(socket.AF_INET, sockets.SOCK_STREAM)
                s = socket.socket()
        Port = 6633
s.connect (('127.0.0.1'))
        print (s.recv(1024).decode())
s.close()
```

9.7 Code of the selection using the flow table

```
#!/usr/bin/env python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/10')

    info( '*** Adding controller\n' )                    //controller setup
    c1=net.addController(name='c1',
                         controller=OVSController,
                         protocol='tcp',
                         port=6633)

    c2=net.addController(name='c2',
                         controller=Controller,
                         protocol='tcp',
                         port=6633)

    c3=net.addController(name='c3',
                         controller=Controller,
                         protocol='tcp',
                         port=6633)

    c4=net.addController(name='c4',
                         controller=Controller,
                         protocol='tcp',
                         port=6633)

    c5=net.addController(name='c5',
                         controller=Controller,
                         protocol='tcp',
                         port=6633)

    c6=net.addController(name='c6',
                         controller=Controller,
                         protocol='tcp',
                         port=6633)
```

```
c7=net.addController(name='c7',
                     controller=Controller,
                     protocol='tcp',
                     port=6633)
c8=net.addController(name='c8',
                     controller=Controller,
                     protocol='tcp',
                     port=6633)

c9=net.addController(name='c9',
                     controller=Controller,
                     protocol='tcp',
                     port=6633)

c10=net.addController(name='c10',
                     controller=Controller,
                     protocol='tcp',
                     port=6633)

c3=net.addController(name='c3',
                     controller=Controller,
                     protocol='tcp',
                     port=6633)

    info( '*** Add switches\n')                                  // switches setup
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')                                     // hosts setup
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)

    info( '*** Add links\n')                                     // network
buildup

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s2').start([])
    net.get('s1').start([])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop(c0)                                                 // when one
```

```
        controller is dead

        if __name__ == '__main__':                    // Assigned role the
        new controller
           setLogLevel( 'c1' )
           myNetwork()

        Return
```

9.8 Controller Fault Tolerance in Mininet


```
{
   "application": {
      "dpctl": "",
      "ipBase": "10.0.0.0/8",
      "netflow": {
         "nflowAddId": "0",
         "nflowTarget": "",
         "nflowTimeout": "600"
      },
      "openFlowVersions": {
         "ovsOf10": "1",
         "ovsOf11": "0",
         "ovsOf12": "0",
         "ovsOf13": "0"
      },
      "sflow": {
         "sflowHeader": "128",
         "sflowPolling": "30",
         "sflowSampling": "400",
         "sflowTarget": ""
      },
      "startCLI": "0",
      "switchType": "ovs",
      "terminalType": "xterm"
   },
   "controllers": [
      {
         "opts": {
            "controllerProtocol": "TCP",
            "controllerType": "ref",
            "hostname": "c1",
            "remoteIP": "127.0.0.2",
            "remotePort": 6633
         },
         "x": "355.0",
         "y": "173.0"
      },
      {
         "opts": {
```

```
            "controllerProtocol": "TCP",
            "controllerType": "ref",
            "hostname": "c0",
            "remoteIP": "127.0.0.1",
            "remotePort": 6633
        },
        "x": "215.0",
        "y": "212.0"
    },
    {
        "opts": {
            "controllerProtocol": "TCP",
            "controllerType": "ref",
            "hostname": "c2",
            "remoteIP": "127.0.0.3",
            "remotePort": 6633
        },
        "x": "496.0",
        "y": "191.0"
    },
    {
        "opts": {
            "controllerProtocol": "TCP",
            "controllerType": "ref",
            "hostname": "c3",
            "remoteIP": "127.0.0.4",
            "remotePort": 6633
        },
        "x": "678.0",
        "y": "236.0"
    }
],
"hosts": [
    {
        "number": "2",
        "opts": {
            "hostname": "h2",
            "IP": "10.0.0.2",
            "nodeNum": 2,
            "sched": "host"
        },
        "x": "687.0",
        "y": "481.0"
    },
    {
        "number": "1",
        "opts": {
            "hostname": "h1",
            "IP": "10.0.0.1",
            "nodeNum": 1,
```

```
            "sched": "host"
         },
         "x": "227.0",
         "y": "485.0"
      }
   ],
   "links": [],
   "switches": [
      {
         "number": "1",
         "opts": {
            "controllers": [],
            "hostname": "s1",
            "nodeNum": 1,
            "switchType": "legacySwitch"
         },
         "x": "347.0",
         "y": "331.0"
      },
      {
         "number": "2",
         "opts": {
            "controllers": [],
            "hostname": "s2",
            "nodeNum": 2,
            "switchType": "legacySwitch"
         },
         "x": "559.0",
         "y": "340.0"
      }
   ],
   "version": "2"
}
End
```


9.9 Dump code in Mininet

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=1411>
<Host h2: h2-eth0:10.0.0.2 pid=1412>
<Host h3: h3-eth0:10.0.0.3 pid=1413>
<Host h4: h4-eth0:10.0.0.4 pid=1414>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None
pid=1417>
<OVSController c0: 127.0.0.1:6633 pid=1403>
mininet>
```

9.10 k-means and k-mediod Clustering in MATLAB

```
rng default; % for repeatability
switcheLocation = [randn(100,2)*0.75+ones(100,2);
   randn(100,2)*0.5-ones(100,2)];
        usageRating =
        ones(length(switcheLocation),1)*1;%abs(randn(length(switcheLocation),1)*1);

        switches = [switcheLocation usageRating];
numberClusters = 10;
```

9.11 Plot the randomly generated switches in MATLAB

```
figure;
plot(switches(:,1),switches(:,2),'.');
        title 'Randomly Generated Switch Locations';
```

9.12 Cluster the switches and return cluster centroids (k-means)

```
opts = statset('Display','final');
[clusterIdx,centroids] = kmeans(switches,numberClusters,'Distance','sqeuclidean',...
   'Replicates',5,' Options', opts);
```

9.13 Plot the controller placements for k-means clustering in MATLAB

```
figure;
hold on
for cluster = 1:numberClusters
   plot(switches(clusterIdx==cluster,1), switches(clusterIdx==cluster,2), '.', 'MarkerSize',12)
end
        plot(centroids(:,1),centroids(:,2),'kx',...
         'MarkerSize',12,'LineWidth',3)
hold off
        Cluster the switches and return cluster mediods (kmediods)
opts = statset('Display','iter');
        [idx,mediods,sumd,d,midx, info] = kmedoids(switches,numberClusters, 'Distance',
'cityblock', 'Options', opts);
```

9.14 Plot the controller placements for k-means clustering in MATLAB

```
figure;
hold on
for cluster = 1:numberClusters
   plot(switches(idx==cluster,1), switches(idx==cluster,2), '.', 'MarkerSize',12)
end
plot(mediods(:,1),mediods(:,2),'kx',...
   'MarkerSize',12,'LineWidth',3)
hold off
```

9.15 Placement Design for the switches topology: in MATLAB

```
rng default; % for repeatability - creates same random data set over and over
        switcheLocation = [randn(100,2)*0.75+ones(100,2);
                randn(100,2)*0.5-ones(100,2)]; % Creates x,y switch coordinates
        usageRating = abs(randn(length(switcheLocation),1)*1); % A matrix that holds
rndomly generated usage data for each switch.
switches = [switcheLocation usageRating];
numberClusters = 10; % How may clusters we want (essentially = to the number of
controllers)
```

9.16 Plot the randomly generated switches (plot our designed topology) in MATLAB

```
plot(switches(:,1),switches(:,2),'.'); % Plots the randomly generated x,y coordinates of all the
switches.
        title 'Randomly Generated Switch Locations';
        Cluster the switches and return cluster centroids (kmeans) - each centroid is the
        optimal placement for the controller
        opts = statset('Display', 'final'); % Creates a holder for all the stats pertaining to each
        randonly initialised kmeans algorithm
[clusterIdx,centroids] = kmeans(switches,numberClusters,'Distance','sqeuclidean',...
         'Replicates',5,' Options', opts); % Call the kmeans algorithm to find the optimal
controller placement for the given data (switches)
                % Out of the 5 randonly initialised instances of the K-means algorithm, the
                % minimum sum of distances (between all points clustered switches and their
        % cebtroid) is 88.2602.
sumOfDistances = [88.2602, 92.37, 89.303, 92, 91]; % The sum of distances pertains to each
instance of the randomly initialised kmeans.
        % The order is random as the initialisation state is randomly selected.
plot(sumOfDistances, '-o')
        Plot the controller centroids for K-means clustering
        figure; % create a new figure
hold on
for cluster = 1:numberClusters % looping through all our clusters
        plot(switches(clusterIdx==cluster,1), switches(clusterIdx==cluster,2), '.',
        'MarkerSize',12) % Plot each cluster
        end
plot(centroids(:,1),centroids(:,2),'kx',...
   'MarkerSize',12,'LineWidth',3) % Plot each centroid
hold off
```

9.17 Plot the controller labels for k-means clustering in MATLAB

```
figure;
hold on
for cluster = 1:numberClusters
   plot(switches(clusterIdx==cluster,1), switches(clusterIdx==cluster,2), '.', 'MarkerSize',12)
end
for centroid = 1:numberClusters
   text(centroids(centroid, 1), centroids(centroid, 2), "C"+string(centroid)) % Labelling each
centroid
```

```
end
hold off
Cluster the switches and return cluster mediods (kmediods)
opts = statset('Display','iter');
[idx,mediods,sumd,d,midx, info] = kmedoids(switches,numberClusters, 'Distance',
'cityblock', 'Options', opts);
```

9.18 Plot the controller placements for K-means clustering in MATLAB

```
hold on
for cluster = 1:numberClusters
   plot(switches(idx==cluster,1), switches(idx==cluster,2), '.', 'MarkerSize',12)
end
plot(mediods(:,1),mediods(:,2),'kx',...
   'MarkerSize',12,'LineWidth',3)
hold off
```

References

[1] Y. Khettab, M. Bagaa, D. L. C. Dutra, T. Taleb, and N. Toumi, "Virtual security as a service for 5G verticals," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, 2018: IEEE, pp. 1-6.

[2] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010, vol. 3, pp. 10-5555.

[3] M. T. Islam, N. Islam, and M. A. Refat, "Node to node performance evaluation through RYU SDN controller," *Wireless Personal Communications*, vol. 112, pp. 555-570, 2020.

[4] T. Jafarian, M. Masdari, A. Ghaffari, and K. Majidzadeh, "A survey and classification of the security anomaly detection mechanisms in software defined networks," *Cluster Computing*, vol. 24, pp. 1235-1253, 2021.

[5] K. Bhushan and B. B. Gupta, "Distributed denial of service (DDoS) attack mitigation in software defined network (SDN)-based cloud computing environment," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, pp. 1985-1997, 2019.

[6] A. Fekih, S. Gaied Fantar, and H. Youssef, "Quality of experience aware replication framework for video streaming in content-centric mobile networks based on SDN architecture," in *Distributed Computing for Emerging Smart Networks: Second International Workshop, DiCES-N 2020, Bizerte, Tunisia, December 18, 2020, Proceedings 2*, 2020: Springer, pp. 75-94.

[7] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli, "Large-scale dynamic controller placement," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 63-76, 2017.

[8] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th International Conference on Emerging networking experiments and technologies*, 2012, pp. 253-264.

[9] O. A. Akanbi, A. Aljaedi, X. Zhou, and A. R. Alharbi, "Fast fail-over technique for distributed controller architecture in software-defined networks," *IEEE Access*, vol. 7, pp. 160718-160737, 2019.

[10] S. Hyun *et al.*, "Interface to network security functions for cloud-based security services," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 171-178, 2018.

[11] A. Fekih, S. G. Fantar, and H. Youssef, "SDN-based replication management framework for CCN networks," in *Web, Artificial Intelligence and Network Applications: Proceedings of the Workshops of the 34th International Conference on Advanced Information Networking and Applications (WAINA-2020)*, 2020: Springer, pp. 83-99.

[12] M. Mbodila, B. Isong, and N. Gasela, "A review of sdn-based controller placement problem," in *2020 2nd International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, 2020: IEEE, pp. 1-7.

[13] S. Farahmandian and D. B. Hoang, "SDS 2: A novel software-defined security service for protecting cloud computing infrastructure," in *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, 2017: IEEE, pp. 1-8.

[14] Z. Guo, S. Zhang, W. Feng, W. Wu, and J. Lan, "Exploring the role of paths for dynamic switch assignment in software-defined networks," *Future Generation Computer Systems*, vol. 107, pp. 238-246, 2020.

[15] B. Yi, X. Wang, K. Li, and M. Huang, "A comprehensive survey of network function virtualization," *Computer Networks,* vol. 133, pp. 212-262, 2018.
[16] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: a survey," *The Journal of Supercomputing,* vol. 71, pp. 3373-3418, 2015.
[17] O. Flauzac, C. González, A. Hachani, and F. Nolot, "SDN based architecture for IoT and improvement of the security," in *2015 IEEE 29th International Conference on advanced information networking and applications workshops*, 2015: IEEE, pp. 688-693.
[18] S. Sujitha, K. P. Priya, and B. Pragathi, "Fault tolerant SDN controller: a survey," *International Journal of Advanced Research*, pp. 186-191, 2016.
[19] M. Karakus and A. Durresi, "Quality of service (QoS) in software defined networking (SDN): A survey," *Journal of Network and Computer Applications,* vol. 80, pp. 200-218, 2017.
[20] S. S. Lee, K.-Y. Li, K. Y. Chan, G.-H. Lai, and Y. C. Chung, "Software-based fast failure recovery for resilient OpenFlow networks," in *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*, 2015: IEEE, pp. 194-200.
[21] A. Rehman, R. L. Aguiar, and J. P. Barraca, "Fault-tolerance in the scope of software-defined networking (sdn)," *IEEE Access,* vol. 7, pp. 124474-124490, 2019.
[22] B. Isong, I. Mathebula, and N. Dladlu, "SDN-SDWSN controller fault tolerance framework for small to medium sized networks," in *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2018: IEEE, pp. 43-51.
[23] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE,* vol. 103, no. 1, pp. 14-76, 2014.
[24] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 55-60.
[25] A. S. Da Silva, P. Smith, A. Mauthe, and A. Schaeffer-Filho, "Resilience support in software-defined networking: A survey," *Computer Networks,* vol. 92, pp. 189-207, 2015.
[26] A. M. D. Tello and M. Abolhasan, "SDN controllers scalability and performance study," in *2019 13th International Conference on signal processing and Communication Systems (ICSPCS)*, 2019: IEEE, pp. 1-10.
[27] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller fault-tolerance in software-defined networking," in *Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research*, 2015, pp. 1-12.
[28] I. Alsmadi and D. Xu, "Security of software defined networks: A survey," *Computers & Security,* vol. 53, pp. 79-108, 2015.
[29] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Communications Surveys & Tutorials,* vol. 20, no. 1, pp. 333-354, 2017.
[30] J. A. Wickboldt, W. P. De Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville, "Software-defined networking: management requirements and challenges," *IEEE Communications Magazine,* vol. 53, no. 1, pp. 278-285, 2015.
[31] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications surveys & tutorials,* vol. 16, no. 3, pp. 1617-1634, 2014.
[32] N. McKeown *et al.*, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review,* vol. 38, no. 2, pp. 69-74, 2008.

[33] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (SD-WAN): Architecture, advances and opportunities," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, 2019: IEEE, pp. 1-9.

[34] M. Mbodila, B. Isong, and N. Gasela, "Towards a Cost-Effective SDN-Enabled on-Demand Security Services Framework," in *2023 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, 2023: IEEE, pp. 1-6.

[35] F. Botelho, A. Bessani, F. M. Ramos, and P. Ferreira, "On the design of practical fault-tolerant SDN controllers," in *2014 third European workshop on software defined networks*, 2014: IEEE, pp. 73-78.

[36] S.-Y. Wang, C.-L. Chou, and C.-M. Yang, "EstiNet OpenFlow network simulator and emulator," *IEEE communications magazine,* vol. 51, no. 9, pp. 110-117, 2013.

[37] B. Agborubere and E. Sanchez-Velazquez, "Openflow communications and tls security in software-defined networks," in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2017: IEEE, pp. 560-566.

[38] F. Alam, I. Katib, and A. S. Alzahrani, "New networking era: Software defined networking," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 3, no. 11, 2013.

[39] S. Shenker, M. Casado, T. Koponen, and N. McKeown, "The future of networking, and the past of protocols," *Open Networking Summit,* vol. 20, no. 1445, p. 1, 2011.

[40] L. Chen and A. Avizienis, "N-version programming: A fault-tolerance approach to reliability of software operation," in *Proc. 8th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-8)*, 1978, vol. 1, pp. 3-9.

[41] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE Communications Surveys & tutorials,* vol. 16, no. 4, pp. 1955-1980, 2014.

[42] A. Lara, A. Kolasani, and B. Ramamurthy, "Simplifying network management using software defined networking and OpenFlow," in *2012 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, 2012: IEEE, pp. 24-29.

[43] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, "A survey on software defined networking with multiple controllers," *Journal of Network and Computer Applications,* vol. 103, pp. 101-118, 2018.

[44] S. S. Lee, K.-Y. Li, K.-Y. Chan, G.-H. Lai, and Y.-C. Chung, "Path layout planning and software based fast failure detection in survivable OpenFlow networks," in *2014 10th International Conference on the Design of Reliable Communication Networks (DRCN)*, 2014: IEEE, pp. 1-8.

[45] S. Math, P. Tam, D.-Y. Kim, and S. Kim, "Intelligent Offloading Decision and Resource Allocations Schemes Based on RNN/DQN for Reliability Assurance in Software-Defined Massive Machine-Type Communications," *Security and Communication Networks,* vol. 2022, 2022.

[46] E. S. Spalla *et al.*, "AR2C2: Actively replicated controllers for SDN resilient control plane," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, 2016: IEEE, pp. 189-196.

[47] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials,* vol. 17, no. 1, pp. 27-51, 2014.

[48] Y. Yu *et al.*, "Fault management in software-defined networking: A survey," *IEEE Communications Surveys & Tutorials,* vol. 21, no. 1, pp. 349-392, 2018.
[49] J. Chen, J. Chen, F. Xu, M. Yin, and W. Zhang, "When software defined networks meet fault tolerance: a survey," in *Algorithms and Architectures for Parallel Processing: 15th International Conference, ICA3PP 2015, Zhangjiajie, China, November 18-20, 2015, Proceedings, Part III 15*, 2015: Springer, pp. 351-368.
[50] M. Bano, A. Qayyum, R. N. B. Rais, and S. S. A. Gilani, "Soft-mesh: a robust routing architecture for hybrid SDN and wireless mesh networks," *IEEE Access,* vol. 9, pp. 87715-87730, 2021.
[51] P. C. Fonseca and E. S. Mota, "A survey on fault management in software-defined networks," *IEEE Communications Surveys & Tutorials,* vol. 19, no. 4, pp. 2284-2321, 2017.
[52] T. Das, V. Sridharan, and M. Gurusamy, "A survey on controller placement in SDN," *IEEE Communications Surveys & tutorials,* vol. 22, no. 1, pp. 472-503, 2019.
[53] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications surveys & tutorials,* vol. 18, no. 1, pp. 236-262, 2015.
[54] A. Jalili, M. Keshtgari, R. Akbari, and R. Javidan, "Multi criteria analysis of controller placement problem in software defined networks," *Computer Communications*, vol. 133, pp. 115-128, 2019.
[55] B. P. R. Killi and S. V. Rao, "On placement of hypervisors and controllers in virtualized software defined network," *IEEE Transactions on Network and Service Management,* vol. 15, no. 2, pp. 840-853, 2018.
[56] A. K. Singh and S. Srivastava, "A survey and classification of controller placement problem in SDN," *International Journal of Network Management,* vol. 28, no. 3, p. e2018, 2018.
[57] J. Lu, Z. Zhang, T. Hu, P. Yi, and J. Lan, "A survey of controller placement problem in software-defined networking," *IEEE Access*, vol. 7, pp. 24290-24307, 2019.
[58] M. F. Bari *et al.*, "Dynamic controller provisioning in software defined networks," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, 2013: IEEE, pp. 18-25.
[59] B. P. R. Killi and S. V. Rao, "Towards improving resilience of controller placement with minimum backup capacity in software defined networks," *Computer Networks,* vol. 149, pp. 102-114, 2019.
[60] G. Wang, Y. Zhao, J. Huang, and W. Wang, "The controller placement problem in software defined networking: A survey," *IEEE Network,* vol. 31, no. 5, pp. 21-27, 2017.
[61] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 344-355, 2017.
[62] M. Gupta, J. Sommers, and P. Barford, "Fast, accurate simulation for SDN prototyping," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 31-36.
[63] M. Hussain, N. Shah, R. Amin, S. S. Alshamrani, A. Alotaibi, and S. M. Raza, "Software-defined networking: Categories, analysis, and future directions," *Sensors,* vol. 22, no. 15, p. 5551, 2022.
[64] O. Friha, M. A. Ferrag, L. Shu, L. Maglaras, and X. Wang, "Internet of things for the future of smart agriculture: A comprehensive survey of emerging technologies," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 4, pp. 718-752, 2021.
[65] P. Manso, J. Moura, and C. Serrão, "SDN-based intrusion detection system for early detection and mitigation of DDoS attacks," *Information,* vol. 10, no. 3, p. 106, 2019.

[66] M. D. J. Bora and D. A. K. Gupta, "Effect of different distance measures on the performance of K-means algorithm: an experimental study in Matlab," *arXiv preprint arXiv:1405.7471*, 2014.
[67] V. Goebel and T. Plagemann, "REsearch/Scientific Methods in Computer Science," *Department of Informatics, University of Oslo*, 2015.
[68] A. R. Roy, M. F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba, "Design and management of dot: A distributed OpenFlow testbed," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014: IEEE, pp. 1-9.
[69] A. V. Aho and J. D. Ullman, *Foundations of computer science*. WH Freeman & Co., 1994.
[70] R. J. Wieringa, *Design science methodology for information systems and software engineering*. Springer, 2014.
[71] R. L. Baskerville, M. Kaul, and V. C. Storey, "Genres of inquiry in design-science research," *MIS Quarterly*, vol. 39, no. 3, pp. 541-564, 2015.
[72] A. Dresch, D. P. Lacerda, J. A. V. Antunes Jr, A. Dresch, D. P. Lacerda, and J. A. V. Antunes, *Design science research*. Springer, 2015.
[73] P. Žukauskas, J. Vveinhardt, and R. Andriukaitienė, "Regional tendencies of corporate social Responsibility," in *Management Culture and Corporate Social Responsibility*: IntechOpen New York, NY, 2018.
[74] A. Dresch, D. P. Lacerda, and P. A. C. Miguel, "A distinctive analysis of case study, action research and design science research," *Revista brasileira de gestão de negócios*, vol. 17, pp. 1116-1133, 2015.
[75] S. T. March and V. C. Storey, "Design science in the information systems discipline: an introduction to the special issue on design science research," *MIS Quarterly*, pp. 725-730, 2008.
[76] V. Machaka and T. Balan, "Investigating Proactive Digital Forensics Leveraging Adversary Emulation," *Applied Sciences*, vol. 12, no. 18, p. 9077, 2022.
[77] J. Iivari and J. R. Venable, "Action research and design science research-Seemingly similar but decisively dissimilar," 2009.
[78] J. E. v. Aken, "Management research based on the paradigm of the design sciences: the quest for field-tested and grounded technological rules," *Journal of Management Studies*, vol. 41, no. 2, pp. 219-246, 2004.
[79] L. M. Kelly and M. Cordeiro, "Three principles of pragmatism for research on organizational processes," *Methodological innovations*, vol. 13, no. 2, p. 2059799120937242, 2020.
[80] E. Van Burg, A. G. L. Romme, V. A. Gilsing, and I. M. Reymen, "Creating university spin-offs: a science-based design perspective," *Journal of Product Innovation Management*, vol. 25, no. 2, pp. 114-128, 2008.
[81] A. R. Hevner, "A three cycle view of design science research," *Scandinavian Journal of information systems*, vol. 19, no. 2, p. 4, 2007.
[82] M. M. Helms and J. Nixon, "Exploring SWOT analysis–where are we now? A review of academic research from the last decade," *Journal of strategy and management*, vol. 3, no. 3, pp. 215-251, 2010.
[83] J. Grenha Teixeira, L. Patrício, K.-H. Huang, R. P. Fisk, L. Nóbrega, and L. Constantine, "The MINDS method: integrating management and interaction design perspectives for service design," *Journal of Service Research*, vol. 20, no. 3, pp. 240-258, 2017.
[84] L. V. Lapão, M. M. da Silva, and J. Gregório, "Implementing an online pharmaceutical service using design science research," *BMC Medical Informatics and decision making*, vol. 17, pp. 1-14, 2017.

[85] P. O. Oviroh, R. Akbarzadeh, D. Pan, R. A. M. Coetzee, and T.-C. Jen, "New development of atomic layer deposition: processes, methods and applications," *Science and technology of advanced materials,* vol. 20, no. 1, pp. 465-496, 2019.
[86] J. W. Creswell, *A concise introduction to mixed methods research*. SAGE publications, 2021.
[87] N. K. Denzin and Y. S. Lincoln, "The discipline and practice of qualitative research," *Handbook of qualitative research,* vol. 2, no. 1, pp. 1-20, 2000.
[88] I. Aliaga, A. Rubio, and E. Sanchez, "Experimental quantitative comparison of different control architectures for master-slave teleoperation," *IEEE Transactions on Control Systems Technology,* vol. 12, no. 1, pp. 2-11, 2004.
[89] O. S. Consortium, "OpenFlow switch specification version 1.0. 0," *http://www. openflowswitch. org/documents/openflow-spec-v1. 0.0. pdf,* 2009.
[90] W. Bekri, R. Jmal, and L. Chaari Fourati, "Internet of things management based on software defined networking: a survey," *International Journal of Wireless Information Networks,* vol. 27, no. 3, pp. 385-410, 2020.
[91] K. N. Sharma, "A Scalable and Fault Tolerant OpenFlow Controller," University of Waikato, 2015.
[92] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," *ACM SIGCOMM computer communication review,* vol. 43, no. 4, pp. 7-12, 2013.
[93] M. Tanha, D. Sajjadi, R. Ruby, and J. Pan, "Capacity-aware and delay-guaranteed resilient controller placement for software-defined WANs," *IEEE Transactions on Network and Service Management,* vol. 15, no. 3, pp. 991-1005, 2018.
[94] L. Hubert, P. Arabie, and J. Meulman, "Multidimensional Scaling in the City-Block Metric: L1 and L2-Norm Optimization Methods Using MATLAB."
[95] R. K. Das, F. H. Pohrmen, A. K. Maji, and G. Saha, "FT-SDN: a fault-tolerant distributed architecture for software defined network," *Wireless personal communications,* vol. 114, pp. 1045-1066, 2020.
[96] T. Wang, H. Chen, G. Cheng, and Y. Lu, "SDNManager: A safeguard architecture for SDN DoS attacks based on bandwidth prediction," *Security and Communication Networks,* vol. 2018, pp. 1-16, 2018.
[97] A. V. Aho and J. D. Ullman, *Foundations of computer science*. Computer Science Press, Inc., 1992.
[98] T. Hu, P. Yi, Z. Guo, J. Lan, and Y. Hu, "Dynamic slave controller assignment for enhancing control plane robustness in software-defined networks," *Future Generation Computer Systems,* vol. 95, pp. 681-693, 2019.
[99] B. Isyaku, K. B. A. Bakar, F. A. Ghaleb, and A. Al-Nahari, "Dynamic routing and failure recovery approaches for efficient resource utilization in OpenFlow-SDN: a survey," *IEEE Access,* vol. 10, pp. 121791-121815, 2022.
[100] T. Hu, P. Yi, J. Lan, Y. Hu, and P. Sun, "FTLink: Efficient and flexible link fault tolerance scheme for data plane in Software-Defined Networking," *Future Generation Computer Systems,* vol. 111, pp. 381-400, 2020.
[101] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "Fattire: Declarative fault tolerance for software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 109-114.
[102] O. Lemeshko, A. Mersni, O. Yeremenko, S. O. Omowumi, V. Volotka, and A. M. Al-Dulaimi, "Application prospects of first hop redundancy protocols for fault-tolerant SDN controllers: a survey," in *2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T)*, 2021: IEEE, pp. 416-420.

[103] Q. Li, Y. Liu, Z. Zhu, H. Li, and Y. Jiang, "BOND: Flexible failure recovery in software defined networks," *Computer Networks*, vol. 149, pp. 1-12, 2019.
[104] A. Malik, B. Aziz, M. Adda, and C.-H. Ke, "Smart routing: Towards proactive fault handling of software-defined networks," *Computer Networks*, vol. 170, p. 107104, 2020.
[105] A. Mantas and F. Ramos, "Rama: Controller fault tolerance in software-defined networking made practical," *arXiv preprint arXiv:1902.01669*, 2019.
[106] J. Teixeira, G. Antichi, D. Adami, A. Del Chiaro, S. Giordano, and A. Santos, "Datacenter in a box: Test your SDN cloud-datacenter controller at home," in *2013 Second European Workshop on Software Defined Networks*, 2013: IEEE, pp. 99-104.
[107] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, p. 106984, 2020.
[108] G. Liang and K. Kökten, "On diagnosis of forwarding plane via static forwarding rules in software defined networks," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, 2014: IEEE, pp. 1716-1724.
[109] Y. Kim, S. M. Raza, D. T. Nguyen, S. Jeon, and H. Choo, "Towards on-demand mobility management in SDN," in *Proceedings of the 12th International Conference on ubiquitous information management and communication*, 2018, pp. 1-5.
[110] L.-D. Chou, C.-W. Tseng, Y.-K. Huang, K.-C. Chen, T.-F. Ou, and C.-K. Yen, "A security service on-demand architecture in SDN," in *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, 2016: IEEE, pp. 287-291.
[111] X. Wang, Z. Liu, J. Li, B. Yang, and Y. Qi, "Tualatin: Towards network security service provision in cloud datacenters," in *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, 2014: IEEE, pp. 1-8.
[112] C.-W. Wu, C.-W. Tseng, W.-Y. Chen, L.-F. Wu, S.-C. Hsu, and S.-W. Yu, "On-demand service function chain based on ipv6 segment routing," in *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2021: IEEE, pp. 336-341.
[113] F. Li, C.-T. Huang, J. Huang, and W. Peng, "Feedback-based smartphone strategic sampling for BYOD security," in *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, 2014: IEEE, pp. 1-8.
[114] J. Kim, M. D. Firoozjaei, J. P. Jeong, H. Kim, and J.-S. Park, "SDN-based security services using interface to network security functions," in *2015 International Conference on Information and Communication Technology Convergence (ICTC)*, 2015: IEEE, pp. 526-529.
[115] S. Zavrak, and M. Iskefiyeli, "A feature-based comparison of SDN emulation and simulation tools", In Proc. International Conference Engineering Technologies (ICENTE), 2017, pp. 214-217.

NORTH-WEST UNIVERSITY
NOORDWES-UNIVERSITEIT
YUNIBESITI YA BOKONE-BOPHIRIMA

7.1.11.1.7.2 PERMISSION TO SUBMIT FINAL COPIES TO HIGHER DEGREES AND PUBLISH FINAL COPIES ON THE NWU REPOSITORY

(for signing off by the supervisor/promoter and for some faculties also the internal examiner)

Graduate:	MUNIENGE MBODILA	Student number:	24087467
Qualification:	PhD	Faculty:	Faculty of Natural & Agricult

1. PERMISSION TO SUBMIT FINAL COPY TO HIGHER DEGREES (and permission to bind final copies for personal use and distribution).

In accordance with the Statute of North-West University the undersigned declares the following with regards to the above-mentioned candidate.

The following title, for the final copy's publication to the NWU repository for dissertations and theses, corresponds exactly with the approved and registered title:

Design and implementation of SDN fault-tolerant framework and security services on-demand

The final copies (electronic and hard copy) are in order and meet the specifications as set out by the relevant policies of the NWU as listed in the Manual for M and D studies;

Amendments as indicated in the examiner's reports have been made to the satisfaction of the supervisor/promoter;

The final copy of the graduate is ready for binding.

The key words as published in the final copy:

: Software-defined networking, fault tolerance, OpenFlow, Ryu, controller placement, security se

2. PERMISSION TO PUBLISH ON NWU REPOSITORY (BOLOKA)

The (mini-)dissertation/thesis may be made available on the NWU repository (Boloka).

Signature of student:	Signature of Supervisor/Promoter:	Signature of internal examiner if required by your faculty.
MUNIENGE Digitally signed by MUNIENGE Date: 2024.07.11 14:58:48 +02'00'	Prof. Bassey Isong Digitally signed by Prof. Bassey Isong Date: 2024.07.11 15:04:36 +02'00'	

7.1.11.1.7.2 EMBARGO REGARDING THE PUBLISHING OF FINAL COPIES ON THE NWU REPOSITORY

(for signing off by the supervisor/promoter and student)

Graduate:	MUNIENGE MBODILA	Student number:	24087467
Qualification:	PhD	Faculty:	Faculty of Natural & Agricu

This Thesis may not be made available on the NWU repository for dissertations.

Please specify the embargo period:

NRF funded students must publish to Select embargo period after the student's graduation ceremony

http://www.nrf.ac.za/nrf_funded_thesis_dissertation_requirements .

The embargo form is signed by the supervisor and student, giving a directive to the Library to withhold the final copy of the mini-dissertation/dissertation/thesis of a student from being made available to the public domain for the period selected above. The form should be submitted with the final copy to Higher Degrees.

Title of the study:

Design and implementation of SDN fault-tolerant framework and security services on-demand

The key words as published in the final copy:

: Software-defined networking, fault tolerance, OpenFlow, Ryu, controller placement, security s

Signature of student:

MUNIENGE Digitally signed by MUNIENGE Date: 2024.07.11 14:50:06 +02'00'

Signature of Supervisor/Promoter:

Prof. Bassey Isong Digitally signed by Prof. Bassey Isong Date: 2024.07.11 15:04:03 +02'00'

Original details: (10512187) M:\HDA Toolbox\Forms\Pdf fillable forms\15.Embargo form for final copies 2021-2022.pdf\ 3 December 2021
File reference: 7.1.11.1.7.2