M060070582

NWU LIBRARY

Software defined wireless sensors network fault tolerance framework

RI Mathebula

orcid.org/ 0000-0002-6313-1003

Dissertation submitted in fulfilment of the requirements for the degree *Master of Science in Computer Science* at the North-West University

Supervisor: Dr B Isong

Co-Supervisor: Ms N Dladlu

LIBRARY
MAFIKENG CAMPUS
CALL NO.:
2018 -11- 14
ACC.NO.:
NORTH-WEST UNIVERSITY

Graduation May 2018

Student number: 28295854

NWU®
NORTH-WEST UNIVERSITY
NOORDWES-UNIVERSITEIT
UNIBESITI YA BOKONE-BOPHIRIMA

Declaration

I, **RISIMATI ISHMAEL MATHEBULA** hereby declare that this research entitled "**Software defined wireless sensors network fault tolerance framework**" is my work and has never been presented for award of any degree in any university. All sources of information used have been duly acknowledged both in text and in references.

Signature: _____ Date: _____

Risimati Ishmael Mathebula

APPROVAL:

Signature.....

Date.....

Supervisor: **Dr. B. Isong**
Department of Computer Science
North-West University
Mafikeng Campus
South Africa.

Signature.....

Date.....

Co-Supervisor: **Ms. N. Dladlu**
Department of Computer Science
North-West University
Mafikeng Campus
South Africa.

NWU LIBRARY

Declaration

I, **RISIMATI ISHMAEL MATHEBULA** hereby declare that this research entitled "**Software defined wireless sensors network fault tolerance framework**" is my work and has never been presented for award of any degree in any university. All sources of information used have been duly acknowledged both in text and in references.

Signature: _____ Date: _____

Risimati Ishmael Mathebula

APPROVAL:

Signature.....

Date.....

Supervisor: **Dr. B. Isong**
Department of Computer Science
North-West University
Mafikeng Campus
South Africa.

Signature.....

Date.....

Co-Supervisor: **Ms. N. Dladlu**
Department of Computer Science
North-West University
Mafikeng Campus
South Africa.

Dedication

I dedicate this research to my lovely family:

My wife Fortunate Mathebula

My son Exousia Mathebula

My dad H.F Mathebula

My mom K.W Mahomane,

My two sisters Mary and Deborah Mathebula

My Mentor Rakhee Ramgolam

Most of all I would like to dedicate this research to my big brother Edwin Baloyi and my company (ARMSCOR) for permitting me to embark on this degree.

Acknowledgements

First of all, I would like to appreciate and acknowledge God for the mighty favour and the blessings that He has shown me, I thank him for giving me the Spirit of Wisdom and Understanding, the Spirit of Counsel and Might, the Spirit of knowledge, Power and of a sound mind that made me to complete my studies.

I would like to also acknowledge my supervisors Dr. B Isong and MS. N Dladlu for their mentorship. They were patient with me and motivated me to work towards the finish line. I am grateful for their support and their supervision during the course of my master's research project, without their assistance my research project was going to be extremely difficult to manage. Dr. B Isong and MS. N Dladlu, thank you so much for your contribution.

My gratitude also goes to my wife and my parents for their unwavering support ever shown to me. Fortu, Mom and Dad you have been there for me through thick and thin. Mom and Dad you advised me, mentored and rebuked me into the person that I am today, thank you. Last but not least I would like to thank my Spiritual father, Apostle S.E Phakathi who has been praying for me. Thank you all amen.

Definition of Concepts

Control Plane: The control plane is the central intelligence of the entire SDN network that manages, monitors and controls the network and provides a global view of the network with the capability of providing advanced performance on fault management, and other standard protocols.

Checkpoint: Checkpoint is defined as the process of taking a snapshot of the network state in order to use the latest state during fault recovery.

Data Plane: Data plane is responsible for forwarding traffic to a selected destination (i.e. switches, wireless sensor, etc.) and switches can either be reliant on the controller or make decisions on their own.

Failure: Failure is defined as the state in which the controller fails to achieve its desired objective, in other words the failure is viewed as the reverse of achieving the required goal.

Fault: A fault is any defect that causes malfunction or outage of the system

Fault Tolerance: Failure is when a system cannot work at all, it cannot perform any function. It's knocked out.

Framework: A framework is a hypothetical description of an essential structure of a FT SDWSN controller that simplifies the complex process and entity of the controller.

Mobile Agent: is an autonomous piece of code that has the ability to move from one node to another in a network in order to perform a specific task.

Network Management System: NMS is a set of software and hardware tools network that enables network administrators to manage and control components of network management framework.

Network Partition: Network partition is a network device that occurs when the network can be split.

OpenFlow: An open protocol that offers a network controller the flexibility to define the connectivity and packets flow across a software defined network.

Software Defined Network: Software defined networking is the separation of the forwarding (data) plane and the control plane and employs open protocols to give the SDN controller a global view of the network and allow network programmability.

Wireless Sensor Network: is a computer network that is composed of a group of wireless sensors that has the capability to monitor, record conditions in a given environment and communicate results to the base station.

List of Acronyms

API:	Application Programming Interface
BS:	Base Station
CSMA:	Carrier Sense multiple access
CTR:	Checkpoint Transfer Request
DR:	Disaster Recovery
EC:	Energy Consumption
EOP:	Exactly Once Property
FATOMAS:	Fault Tolerant Mobile Agent System
FT:	Fault Tolerance
FTE:	Fault Tolerance Enabler
FTF:	Fault Tolerance Framework
FTM:	Fault Tolerance Manager
FTPF:	Fault Tolerance Path Finding
HA:	Home Agent
HB:	Heartbeat Manager
IoT:	Internet of Things Technology
LgA:	Log Agent
LR-WPAN:	Low-Rate Wireless Personal Area Network
LTR:	Log Transfer Request
MA:	Mobile Agent
MAFT:	Mobile Agent Fault Tolerance
MAG:	Multi Agent Group
MH:	Mobile Hosts
MSS:	Mobile Service Stations
NAP:	Norwegian Army Protocol
NMS:	Network Management System
NFV:	Network Function Virtualization
ODL:	Open Day Light
QoS:	Quality of service
RBP:	Relative Blocking Probability
SDN:	Software Defined Network
SDWSN:	Software Defined Wireless Sensor Network

TDMA:	Time Division Multiple Access
WSN:	Wireless Sensor Network
TCP/IP:	Transmission Control Protocol/Internet Protocol
IDs:	Identities
RO:	Research Objective
MEMS:	Micro-Electro-Mechanical-Systems
EOP:	Exactly Once Property (EOP)

Table of Contents

Dedication ii

Acknowledgements iii

Definition of Concepts iv

List of Acronyms v

Table of Contents vii

List of Figures x

List of Tables xii

ABSTRACT xiii

CHAPTER 1 1

Introduction and Background 1

1.1 Introduction 1

1.2 Background Information 4

1.3 Problem Statement 6

1.4 Research Questions 7

1.5 Research Aim 7

1.6 Research Objectives 8

1.7 Research Justification 8

1.8 Research Scope 8

1.9 Research Methodology 8

1.9.1 Design a 3-phase SDWSN controller's framework 9

1.9.2 Analysis and evaluation of the proposed framework 10

1.10 Structure of Dissertation 10

1.11 Chapter Summary 11

CHAPTER 2 11

Review of the Literature 11

2. Chapter Outline 11

2.1. Overview of Wireless Sensor Network 12

2.2. WSN Benefits and Applications 14

2.3. WSN Challenges 15
2.4. Overview of Software Defined Networks 18
2.5. Software Defined Network Benefits and Application 20
2.6. SDN Architectures 21
2.7. Software Defined Wireless Sensor Network 22
2.8. SDWSN Advantages and Applications 25
2.9. Fault Tolerance Techniques 26
2.10. Mobile Agent Fault Tolerance 28
2.11. MAFT Challenges 28
2.12. Related works on SDWSN and controller Fault Tolerance 30
2.13. SDWSN and Traditional WSN Comparison 32
2.14. Chapter Summary 34
CHAPTER 3 35
Analysis of Fault Tolerance Approaches and Mechanisms 35
3.1 Chapter Outline 35
3.2 Overview 35
3.3 Methodology 35
3.4 Evaluation Criteria 36
3.5 Analysis of Mobile Agent Fault Tolerance Techniques 38
3.6 Discussion 48
3.7 Chapter Summary 49
CHAPTER 4 53
SDWSN Fault Tolerance Framework 53
4.1 Chapter Outline 53
4.2 Overview 53
4.2 Choice of FT Approach 54
4.3 The SDWSN Fault Tolerance Framework 55
4.3.1 Types of Faults to Detect 55
4.3.2 Fault Tolerant Execution Cycle 56
4.4 Proposed FT Strategy Design 58
4.4.1 3-Phase Controllers 58
4.4.2 Fault Tolerance Manager Framework 60

4.5 Fault Detection and Fault Recovery 66
4.6 Controllers Synchronization and Network State Consistency 71
4.7 Discussion 72
4.8 Chapter Summary 73
CHAPTER 5 74
Framework Evaluation and Results 74
5.1 Chapter Outline 74
5.2 Overview 74
5.3 Simulations 74
5.3.1 Objective 75
5.3.2 Setup and Configuration 75
5.4 Simulations Result 82
5.5 Theoretical Evaluation of SDWSN FT Framework 86
5.6 Discussions 90
5.7 Chapter Summary 90
CHAPTER 6 91
Sumrnary, Conclusions and Future Works 91
6.1 Chapter Overview 91
6.2 General Summary 91
6.3 Conclusions 92
6.4 Recommendation 92
6.5 Future work 93
References 94

List of Figures

Figure 1.1: SDN controller OpenFlow protocol [10] 2
Figure 1.2: Distributed Controller [13] 5
Figure 2.1: Sensor node architecture [28] 12
Figure 2.2: Security Attacks on WSN [37] 16
Figure 2.3: Hidden terminal [28] 18
Figure 2.4: Traditional network and software defined network architecture [10] 19
Figure 2. 5: SDN system design architecture SDN [40] 21
Figure 2. 6: Types of sinks in a WSN [45] 22
Figure 2.7: Architecture module [19] 24
Figure 2. 8: Fault tolerance path finding algorithm [55] 27
Figure 2. 9: Failure model for mobile agents [67] 29
Figure 2. 10: Distributed controller [17] 30
Figure 2. 11: Centralized controller [17] 30
Figure 2. 12: Simulation scenario for a SDWSN and MTE WSN lifetime [79] 33
Figure 3.1: FATOMAS Architecture [80] 39
Figure 3.2: Mobile agent versions [81] 40
Figure 4.1: Cycle for Fault Tolerant Execution [103] 57
Figure 4.2: EDD controllers 58
Figure 4.3: Msg_Mnr and Heartbeat communication 59
Figure 4.4: The FT managers 60
Figure 4.5: The FTM 61
Figure 4.6: Heartbeat manager 62
Figure 4.7: checkpoint and rollback algorithm 64
Figure 4.8: Checkpoint Manager stable storage 65
Figure 4.9: Voting Manager Execution 66
Figure 4.10: Fault detection & recovery cycle 67
Figure 4.11: Fault Detection Pseudocode 68
Figure 4.12 Fault Diagnosis Pseudocode 69
Figure 4.13: Recover fault Detection 70
Figure 4.14: Execute Recovery 70
Figure 4.15: Consistency on the EDD Controllers 71
Figure 4.16: Proposed SDWSN architecture [17] 72

Figure 5. 1: Setting up Distributed ODL Controllers 76
Figure 5. 2: Running ODL Controllers 77
Figure 5. 3: No Devices connected 78
Figure 5. 4: Setting up a network topology 79
Figure 5. 5: Tree network topology 80
Figure 5. 6: Packets received 81
Figure 5. 7: 125 host with defined IP addresses 82
Figure 5. 8: Nodes information 83
Figure 5. 9: Node Id –OpenFlow :12 84
Figure 5. 10: Node Connector statistics for Node Id-OpenFlow: 12 84
Figure 5. 11: flow table statistics for Node Id 85
Figure 5. 12: Flow Table Stats for Node ID – OpenFlow: 12 85

List of Tables

Table 2. 1: Related work on SDWSN and traditional Wireless Sensor Networks [18].....32
Table 3. 1: Analysis of Fault Tolerance Techniques in Mobile Agent Systems.....50
Table 3. 2: Advantage of each FT Technique.....51
Table 3. 3: Disadvantage of each FT technique.....52
Table 5. 1: Theoretical evaluation 87
Table 5. 2: Description of Operations on FTF 88

ABSTRACT

Software-Defined Networking (SDN) is a new network paradigm that supports the decoupling of the control plane and the data plane where the controller is the central intelligence or the brain of the entire network. SDN has attracted widespread application due to its flexibility in networks management and control. SDN has also been adopted in several existing network technologies and wireless sensors network (WSN) is not an exception resulting to SDWSN. At the moments the SDWSN controller is only single and lacks fault tolerance (FT) capability. As the brain of the network, it is prone to several faults, errors, security attacks, and poses as a single point of failure of the network. Therefore, to ensure the single point of failure is eliminated in the SDWSN controller, this research proposed a FT framework in the SDWSN control plane that ensures continuous reliability and availability of the SDWSN controller's execution. This is achieved by designing a FT framework which is made of two core components: the 3-phase or the EDD controllers and the FT Manager (FTM). On the 3-phase controllers, only one controller is responsible for network management while the other two remain as standby during the event of failure.

Moreover, FTM also monitors the executing controller using heartbeat techniques to detect the occurrence of fault or failure. The FTM is made of five managers: heartbeat, execution, checkpoint, diagnostic and voting. The managers operate cooperatively to ensure the reliability, availability, state consistency and better performance of the SDWSN controller. We also present the design of each component, its operation and discussion. In addition, simulations and theoretical evaluations were performed on the FT framework as proof of concept. The simulations utilized the OpenDayLight (ODL) controllers and the Mininet software tool while the theoretical evaluations were carried out based on several criteria and parameters.

The results obtained from the simulation show that the SDWSN controller is indeed the brain of the entire network and is capable of providing an abstract or global view of the entire network. In terms of the theoretical evaluation, it shows that FT framework is standardized and is effective in ensuring the controller's reliable execution in the detection and recovery from faults or failures in the network. The results show that our FT framework can offer effective fault detection and fault recovery to ensure the SDWSN controller remains operational in the face of faults or failure in the network. Moreover, if the FT framework is adopted for implementation in the real-world SDWSN, it could go a long way to eliminate the single point of failure posed by a single controller in OpenFlow architecture.

Keywords: *SDN, WSN, SDWSN, Controller, FT, FTM, Faults, Failure, Framework, Detection, Recovery, Mobile Agents.*

CHAPTER 1

Introduction and Background

1.1 Introduction

In recent years, the network paradigm called wireless sensor networks (WSNs) has received significant attention in terms of development and deployment. Today, WSN is gaining momentum as a result of its limitless potential and has been known as the building block for Internet of Things (IoT) technology [1]. WSN is a network characterized by the self-organizing joint work of several numbers of distributed sensors and associated devices which are used for sensing and processing wireless data communication within a short distance [2]. The network is simple to deploy and it is categorized by its unique ambiguous nature. In addition, WSN support extensive numbers of applications which include sensing indoor, environment and traffic monitoring, military defense, disaster relief, biological applications and so on [3]. The applications can be utilized remotely via several small sensor nodes that are assembled with the wireless transmitters coupled with data collection and processing capability. These sensor nodes can be used to sense temperature, and pressure, and to exchange information with the close nodes or with nodes outside their environment using ad hoc networking [4]. In the realm of WSN, sensors also consist of dominant nodes known as Gateway nodes or Base Stations (BS) to which other sensor nodes communicate events. In general, sensor nodes can be deployed anywhere, even in places not accessible to humans, in order to sense and detect events. Due to network operation of the sensor nodes, sensor nodes offer benefits which are considered advantageous over traditional sensors. However, to operate effectively, they have to be more reliable and dependable [5]. Consequently, WSN is not without challenges. With the distributed and the heterogeneous nature of today's networks, the present protocols of the traditional networks find it difficult to cope with the complex challenges of managing the network effectively and efficiently. Thus, to deal with the complexities, a technique that can manage network resources flexibly and dynamically was developed. The new network technology is the Software Defined Networking (SDN) [6].

SDN is a network paradigm developed to cope with the inherent limitations encountered in the traditional network management and its lack of flexibility [6], [7]. Its developmental goals were to simplify innovation and the programmability of the network management and control.

The paradigm of SDN operates with the architecture that separates network control from its forwarding. It enables flexibility on the network control and management which in turn brings

about network virtualization, new protocol deployment and new network management strategy [7]. Today, a few SDN protocol exist and among them the commonest is called the OpenFlow [8] (see Figure 1.1). In the OpenFlow architecture, SDN is centered on building a computer network by separating it into three systems namely; the control, applications and data planes [9]. In particular, the controller plane provides advanced performance on fault management, NetFlows and other standard protocols [8]. It typically handles configuration and management of the SDN compliant devices and understands the network topology. SDN operates and relies on the availability of one controller which is embedded with the capabilities of defining data paths, updating the policies dynamically, traffic forwarding decisions, faults detections, process, connect and request based on desired requirements, for example if the client requires Quality of Services (QoS), and even performing link management between devices [9]. For instance, on the data plane, it is responsible for forwarding traffic to a selected destination and switches can either be reliant on the controller or make decisions on their own.

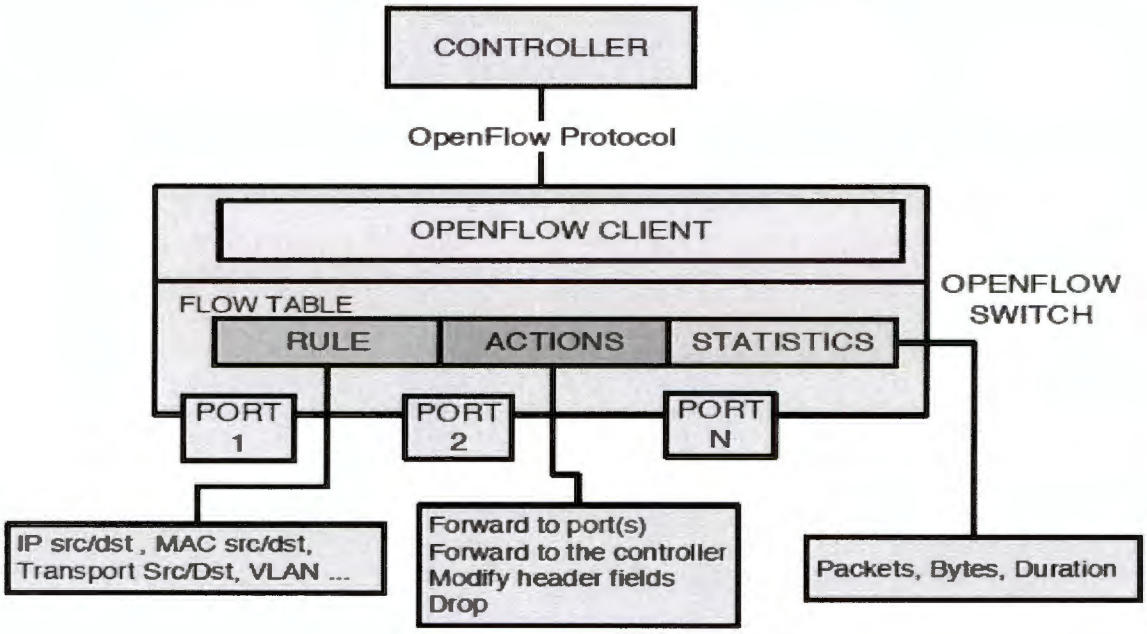


Figure 1.1: SDN controller OpenFlow protocol [10].

In SDN, with the presence of the centralized software controller, several network management functions can be handled effectively and optimally. In other words, the SDN controller is the central intelligence of the entire network. SDN focuses on the concept of setting logical functions of high flexibility over the physical infrastructure, according to the application needs thereby freeing network administrators from the complex tasks of network services management, but via low level functionality abstraction. SDN is gaining momentum, and a lot of efforts have been devoted to improving it. Moreover, being a new technology, there are several challenges it faces such as security, switch design issues, controller placement,

reliability (e.g. fault tolerance), dependability and so on [11]. In addition, it has also attracted widespread applications such as in cloud computing, wireless networks, and so on [8].

In the perspective of wireless networks application, WSN is one of the multi-hop wireless developments adopted by the SDN [12]. Its adoption was due to the growing complexity of managing modern wireless networks, which calls for a flexible approach, that is, utilizing the benefits of SDN to manage and control WSN, otherwise known as SDWSN. The implementation of SDWSN offers huge benefits especially in the capacity of node and resource management [13]. In SDWSN, the controller is obligated to intelligently manage and control the network and provide abstract and global view of the network, and the capability to detect and monitor faults in the network. Consequently, the SDWSN controller is capable of determining optimal routes to save energy as to well as determine sensors node failure. However, the SDWSN sensor nodes are deployed in the unattended environment or in the field to gather required information. These sensors are resource constrained and are exposed to a number of security attacks. Moreover, OpenFlow-based SDWSN uses a single controller and based on its criticality in the network, it is always a primary target of attacks and poses a single point of failure to the entire networks. Thus, the SDWSN controller and its associated sensors have to be secured and be fault tolerant at all times [14].

Given the critical context, this research is positioned to improve the reliability of SDWSN. That is, the controller in the OpenFlow-based architecture has to be fault tolerant to eliminate the single point of failure. To achieve this, we will explore the different existing fault tolerance strategies in the literature to identify an optimal approach which is suitable to make the SDWSN controller fault tolerant. Making the controller fault tolerant will ensure the reliability and dependability of the network in the face of faults or failures. Although a traditional SDN controller can handle specific failures, it does not address complex faults, for example if the controller fails, the whole network will shut down [15]. This can be achieved with a 3-phase synchronized controllers with fault tolerance mechanism. The essence is to ensure state consistency of the controller's executions, communications and network global-view[16], which are essential for recovery in the face of failure, which in turn eliminates the chances of a network failure in case of a single controller's failure.

Today, several innovations have been introduced in the capacity of SDN but several challenges remain to make it more reliable, secure and dependable.

1.2 Background Information

The adoption of SDN to WSN introduced the concept and application of SDWSN that enables devices to monitor and control the management of resources. SDWSN is a new fascinating paradigm for LR-WPAN which can be achieved by applying the SDN model into wireless sensor network [17]. The SDN model has been deployed in various enterprise solutions such as network function virtualization (NFV), enterprise networks and data centres. Recently SDWSN has been recognized to fully explore and powerfully apply the resources of WSNs [18]. SDWSN aims to handle different challenges and requirements that traditional WSN cannot solve. It offers quite a lot of opportunities which include better resource reuse mechanism, improved note re-tasking and simplified transition to protocols for implemented WSN [19].

The fast growing development of recent applications of WSNs introduced new challenges such as node failure, energy shortage, security etc., which introduce harmful threats to the entire lifespan of the network [18]. SDN was introduced into WSN in order to enable its ability to efficiently manage current wireless networks and exploit the advantages of SDN to regulate WSNs; this approach is commonly known as SDWSN. The deployment of SDWSN brings a lot of benefits, particularly its capacity to manage resource and flexibility of node.

Accordingly, the SDN controller is obliged to handle effectively the wireless network, create an overall view of the network and handle all kind of errors or faults emerging in the network [12]. Rahmani *et al.* [20] were among the first researches to implement SDN idea to WSN with the objective of applying OpenFlow basics to the sensor nodes. Today, the SDWSN has been recognized for effectiveness in handling WSNs resources, which offers several advantages over the traditional WSN as well as many prototypes introduced and implemented. However, more research is required in order to justify the quality and practicability of using SDWSNs due to several SDWSN challenges that were identified which need in-depth research [8], [12].

In the SDWSN, the controller and the sensor nodes are very important nodes which are required to be secured and reliable in terms of interaction and execution in the network. Thus, fault tolerance is inevitably important because having a single controller, for instance, makes it a single point of failure. The sensors and the SDWSN controller have to be protected at all times such that during any attacks likely to result in failure, both can be available through fault tolerance [14]. The SDWSN controller is proficient in determining optimal routes to save energy as well as determining sensors node failure. As shown in Figure 1.2, sensor nodes are

vulnerable to failure and error because they are deployed in the unattended environment where they collect data. These sensor nodes are regularly placed in large numbers without taking into consideration security and physical access to every single node. Thus, sensors are exposed to a number of attacks [21]. Moreover, having only one SDWSN controller in the OpenFlow-based architecture is a serious challenge especially if the controller is compromised. In this case, the whole network will shut down because the controller is the brain of the entire network. Though incorporating SDN features to the WSNs introduces certain opportunities or advantages, it also introduces several challenges that need to be managed effectively. Technical problems encountered when adopting SDN to WSN were discussed in [12]. Network security and fault tolerance is an already existing challenge in wireless LANs which is also problematic in SDWSN [13].

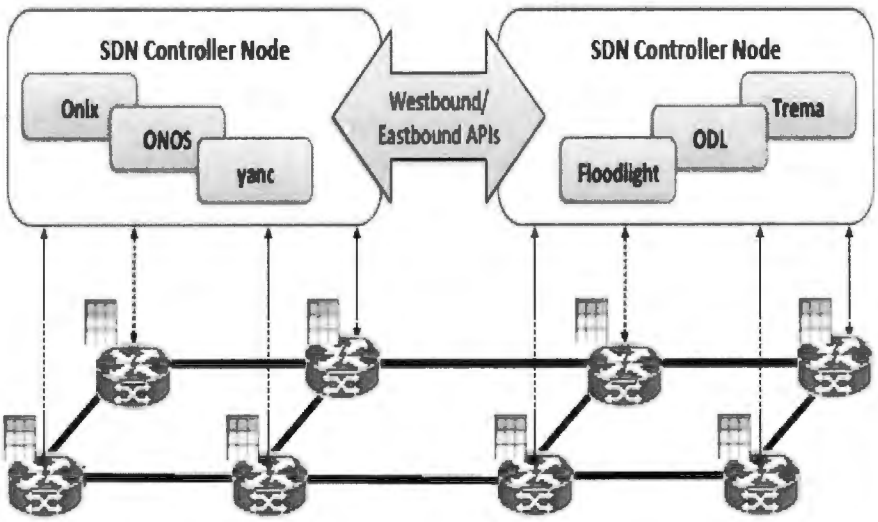


Figure 1.2: Distributed Controller [13]

In Figure 1.2, the SDWSN controller is responsible for handling configuration management of the SDWSN compliant devices and understands the network topology. Such capabilities actually introduce various faults and attract various attacks which present new threats to the network management system for a wireless sensor [8]. SDWSN technology supports OpenFlow protocol [8], [20]. The OpenFlow protocol supports three message types namely; the controller-to-switch, asynchronous and symmetric message, each having multiple subtypes [8]. The controller-to-switch messages are initiated by the controller and are used to directly manage and inspect the state of the switch. On the other hand, asynchronous messages are initiated by the switch and are used to update the controller about the network events and changes to the switch state. Symmetric message can be initiated by either the controller or the

switch and can be sent without solicitation. Nevertheless, these approaches do not solve problems and challenges addressed in [19], such as node failure solution, interruption in communication and limited supply of energy.

In particular, fault tolerance and energy consumption is a major issue in SDWSN as in WSN, because the life cycle of a sensor node is short, which has an influence on the design of sensor network applications and protocols. What is desirable is to introduce an affordable, efficient and reliable way to implement network sensors. Many proposed solutions were discussed in [19]. According to Doherty *et al.* [22] naturally a wireless network is not reliable, and there is a lot of reasons responsible for preventing packets from reaching a receiver. A transmission between different independent transmitters can corrupt each other's signal if it happens that transmission is on the same channel, and this will prompt retransmission, consuming additional energy and time [22]. One of the most important issues concerning split control architectures in SDN is related to its flexible resilience to faults tolerance, which can affect the interaction between data and the control plane [23]. Therefore, there is a need to introduce an effective technique that allows fault tolerance on a SDWSN. Using the SDWSN controller to monitor and manage these emerging faults introduces concern as it reduces the performance of the controller that has other tasks to handle [24]. It is important to introduce a proficient technique or strategies that can be used to solve challenges that are inherited from SDN and WSN in order to manage failure of the entire network and not create an overwhelming resource burden to the controller.

1.3 Problem Statement

SDN is a network technology developed to cope with the natural limitations encountered in the traditional network management and their lack of flexibility [6], [7]. In the perspective of WSN, SDN is used to manage, control and make network management flexible, in particular, the capacity of node and resource management [13]. The SDWSN controller is designed to manage and control the network and provides a global view of the network topology as well as the capability to detect and monitor faults in the WSN. Moreover, the SDWSN controller is capable of determining optimal routes to save energy and determine sensors node failure. However, the issue of reliability remains a serious challenge and has limited the benefits derived from SDWSN in the perspective of the controller.

The SDN using the OpenFlow architecture has a centralized intelligence network control that is specifically managed by the SDN controller. In other words, the SDN controller manages the entire network and serves as the only decision making entity [6], [7]. However, the

centralized property represents a single point of failure. That is, if the controller fails the whole network will collapse. Consequently, the controller is a critical element as well as a potential primary target for attackers or hackers. Thus, it has to be secured and fault tolerant. Furthermore, SDWSN sensor nodes are deployed in the unattended environment and are prone to resource constraints, link failure and security attacks which could lead to the failure of the entire network. With SDWSN, sensors failure can be detected but cannot be completely stopped. Therefore, the reliability of the SDN controller is indispensable. In the case of attack which can lead to network failure, there should be a fault tolerant mechanism in charge of detecting the network [14]. With the controller, though one proposed solution is the use of distributed controllers, there is still a challenge of ensuring consistency and a global view of the network.

Given the critical context, this research is positioned to design a 3-phase SDN controller that has the ability to synchronize its current state and provide a global and consistent view of the network while still being fault tolerant in the event of failure.

1.4 Research Questions

To address the above stated problem, the following research questions (RQs) will be answered:

RQ1: Which existing MAFT approach is more efficient in a resource constraint environment? This question is answered in Chapter 3 with analysis of existing MAFT approaches.

RQ2: How can we design a SDWSN framework which is more reliable based on RQ1?

To answer this question, the following sub-questions are answered:

RQ2.1: How can we make the controller fault tolerant with less resource consumption?

See Chapter 4.

RQ2.2: How can we prevent the controller from being a single point of failure in the face of attack?

See Chapter 4.

RQ3: How can we ensure a global view of the network in the face of a 3-phase controller in the SDWSN? See Chapter 4 and Chapter 5.

1.5 Research Aim

The main aim of this research is to design a fault tolerant framework for SDWSN to ensure reliability and availability of the SDWSN controller in the face of fault or failure.

1.6 Research Objectives

To address the challenges outlined in the problem statement and realize the aim of this research as well as offer answers to the research questions, the following objectives are employed:
RO1: Conduct literature survey on SDWSN and analysis of existing MAFT mechanisms to identify the fault tolerant strategy with least resource constraint capability.

RO2: Design a resourceful fault tolerance framework for SDWSN controller.

RO3: Simulate the proposed fault tolerant controller to evaluate its performance and effectiveness.

1.7 Research Justification

A controller is one of the critical components of a SDWSN that serves as the brain of the network and offers improved communication between the OpenFlow data plane and application plane [9]. Although the controller contains such power functionality, it is prone to errors and faults which can be difficult to recover from. Without a proper mechanism to offer fault tolerance on the controller, it will remain with the challenge of reliability and availability. Thus, it is imperative to introduce a fault tolerance mechanism in the control plane. To achieve this challenge, this research is geared towards proposing and designing an effective fault tolerance system which will assist the controller to have the intelligence to detect and recover faults arising from the network. In this case, we proposed 3-phase controllers and a fault tolerance manager (FTM) to offer fault tolerance control.

1.8 Research Scope

This research will design a fault tolerance framework for OpenFlow-based SDWSN controller. The fault tolerance framework will be based on ideas from various existing fault tolerance strategies. This will be achieved with evaluation of existing techniques to identify approaches that enhance the availability and the reliability of a system. In addition, the proposed approach will be evaluated theoretically.

1.9 Research Methodology

This study will be based on both quantitative and qualitative research approaches, to collect and gather data. The aim of qualitative research is to describe certain aspects of a phenomenon, with a view to explaining the subject of study [25]. We will use various written materials in the form of textbooks, journals, and white papers and will consult experts as our primary sources

of information regarding our topic. The gathered information will be analyzed and the results will be used to propose a conclusion pertaining to the existing fault tolerance mobile agent and SDWSN. The study will consist of two main components i.e. exploratory and data analysis. The exploratory aspect of the study will focus on a comprehensive literature review of FTMA and SDWSN implementations. The data analysis components, on the other hand, will focus on gathering and analysis of the figures from the SDWSN and FTMA, implementations experimental results and simulation. This study mainly relates to evaluation and model based research. The reason for using qualitative research methodology is that the data analysis part in this study is not based on variables that we have control over, rather it is from related experiments [26]. In order to address the research questions, we will follow literature review and comparative data analysis strategies. The literature review process involves key word identification, material location, followed by thorough analysis of the useful documents and material within literature map [26]. A survey of the current literature will be conducted in order to obtain answers to the research questions posed above.

The following steps will be used to facilitate achievement of the research objectives:

1.9.1 Design a 3-phase SDWSN controller's framework

An efficient framework for a 3-phase SDWSN controller will be designed, based on the knowledge gained from the literature review conducted. To design the FT framework, the following will be done (Research objective RO):

RO1: Conduct literature survey on existing MAFT and SDWSN fault management.
We will carry out a literature review on SDWSN and its existing FT techniques with the intention *to:*

i. Determine mechanisms used to manage faults in SDWSN controller.
ii. Determine techniques that can dynamically handle fault tolerance when the controller fails.
iii. Identify significant functionality and components.
iv. Identify advanced techniques and components that can be adopted to handle and support FT on SDWSN.

Furthermore, we will analyze existing MAFT mechanisms to identify a suitable FT strategy for SDWSN controller fault tolerant system. This is important to identify and recommend a resourceful FT strategy that can be adapted to SDWSN.

RO2: Propose and design an appropriate and resourceful FT approach for SDWSN controller. In order to develop and design the proposed FT framework, the knowledge gained in performing the literature review will be utilized. This will involve adapting essential components and techniques of FT from MAFT to SDWSN and outlining the newly adopted components and functions of the suggested model.

1.9.2 Analysis and evaluation of the proposed framework

The proposed framework will be simulated in a SDWSN environment in order to evaluate its performance. The evaluated model will be based on the following parameters:

1) Distributed SDWSN controllers (3-Phase SDWSN controllers).
2) SDWSN Controller Synchronization
3) Fault Tolerance Manager
4) Checkpointing Manager
5) Voting Manager
6) Heartbeat Manager

RO3: *Simulate the proposed fault tolerant controller to evaluate its performance and effectiveness*. With the proposed FT framework, the following suggested technologies (mainly open source software) will be used to evaluate the performances:

i. Virtual Box, Mininet, Open Day Light (ODL) controllers (Distribution-karaf-0.4.3-Beryllium-SR3), Ubuntu server, Putty and JAVA

In addition, these technologies will be used to achieve the following:

ii. Develop a proof of concept by simulating a SDWSN network.
iii. Execute experiments with relatively applicable data sets.

1.10 Structure of Dissertation

The remaining chapters of the thesis will be organized as follows:

Chapter 2: Literature Review

Chapter two is all about collecting, understanding and interpreting literature observations or findings that were done during previous research. The results from previous literature provides the researcher with better guidance to conduct further research, and make decisions based on findings.

Chapter 3: Methodology and Analysis of fault tolerance approaches and mechanisms for mobile agents

This chapter discusses an in-depth comparative analysis of existing Fault Tolerance (FT) strategies and mechanisms that can be used to handle fault tolerance on a network system.

Chapter 4: Software Defined Wireless Sensors Network Fault Tolerance Framework

This chapter introduces the SDWSN FT Framework that describes how fault tolerance will be catered for on the controller. Components of the framework and the strategies are explained in detail in this chapter.

Chapter 5: Framework Evaluation and Results

This chapter presents the analysis and results of the experiments performed in this research, and also evaluates the framework which is introduced in Chapter four.

Chapter 6: Conclusion, Recommendations and Future Work

Lastly chapter six provides a brief summary and conclusion pertaining to the research that was conducted and shows how the aim and objectives were achieved. It also outlines the recommendations and suggestions for future work.

1.11 Chapter Summary

This chapter has discussed the introduction and the background theory of the study. The problem statement was discussed together with the aim and objectives. The research questions were also identified and discussed. The next chapter focuses on the literature review.

CHAPTER 2

Review of the Literature

2. Chapter Outline

The purpose of this chapter is to identify and review past research work and publications on SDWSN area, identify research gaps in the existing literature, and review the existing state of knowledge on the matter. This chapter provides theoretical framework for interpretation and analysis, to be used in designing research instruments, knowledge questioning and generation of required solutions. It also outlines the conceptual framework of the study and also looks at the theories and tools involved in road provision, maintenance and sustainability challenges.

2.1. Overview of Wireless Sensor Network

Wireless sensor networks (WSNs) have gained momentum, due to growth of micro-electro-mechanical systems (MEMS) technology that has played has major role in the development and proliferation of Smart-Sensors (SSs) [27]. As compared to the traditional sensor, Smart-Sensors are intelligent, small and cheap but still contain limited computing and processing resources [5]. These smart-sensors have the ability to measure, to collect information on the environment where they are deployed, sense and forward the collected data to the Base Station. Sensor nodes are the building blocks of a wireless sensor network with an architecture presented on Figure 2.1. A sensor node consists of the following components:

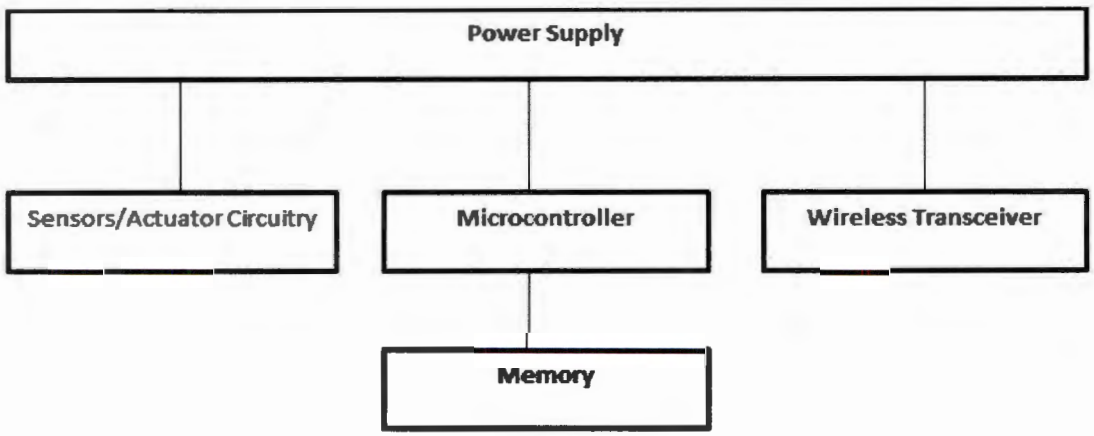


Figure 2.1: Sensor node architecture [28]

a. Power Supply

The sensor node consists of a power supply which is responsible for providing energy to the sensor node. The power supply is in the form of a battery that has a limited energy lifetime, although certain batteries have the ability to recharge themselves if the user is not consuming a lot of energy during the recharging process. But if the sensor consumes a lot of energy when the battery is recharging, the battery will not recharge. According to [28], in most WSN applications it is sometimes impractical to recharge or replace batteries on the sensor node. This implies that if the battery's lifetime ends, the sensor node life time ends as well. For this reseason, a lot of research has been done to find ways to recharge the batteries by absorbing energy such as heat or vibration from the environment to extend the lifespan of the node. Therefore power supply is one of the most important elements of a wireless sensor network [19].

b. Wireless Transceiver

A wireless transceiver carries out modulation, demodulation, reception and transmission of digital data through a wireless channel. In most cases the sensor node relies on the radio frequency channel in unrestricted bands in which few transceivers are available commercially on the market. Sensor nodes transceivers are not selected to enhance packets error or data throughput, however, efficient energy or power consumption is very significant [2]. This is accomplished through low power-design and by selecting reasonably low output energy. Data rates presented by the wireless transceivers are strained to a range of 10 to 100 Kb/s. Most wireless transceivers demand a huge amount of the overall power budget, and based on the combination of the transceiver and microcontroller, the microcontroller can execute several instructions using the same power costs as required for transferring packets to the wireless channel. This is the reason why computation is cheaper as compared to communication. Furthermore, energy costs for receiving and transferring packets are similar due to low power transmission. In most designs, packet transmission has almost the same receiving energy costs [29]. Often the transceiver can hang on channels for packets without receiving anything. This is known as idling. The amount of energy lost during the idling state is equivalent to half or more of the energy used when a packet is received.

c. Sensor or Actuator Circuitry

Sensors or actuators have an electrical device called a transducer which converts one form of energy into another, i.e., it converts it from digital to analog or the other way around. An example of a sensor can be accelerometer, light sensor or temperature sensor. Examples of actuators are a dimmers or speakers.

d. Microcontroller

A microcontroller executes protocol stack and application software. Generally a microcontroller is employed in sensor nodes to enhance consumption energy and not to optimize performance. Most sensor nodes are designed with 8-16 bit processors, similar to the MSP430 Texas Instruments and also similar to the 128 Atmel ATmega by means of clock rates of a small number of 10s of MHz [29]. They often only use a small number of kilobytes of

computer memory (RAM) and also a small number of 10s kilobytes of flash RAM free to save code and data.

e. Memory

The memory component is required to save packets and readings collected by the sensors. Although a RAM can perform fast, the only problem it has is that of losing its state and information in case of power failure [18].

2.2. WSN Benefits and Applications

WSN is an interesting and challenging research area that simplifies emerging applications to communicate with the external environment. It is made up of numerous nodes that gather information from the environment such as building automation, transport, systems army and naval operations, health care, atmospheric condition, and many more [23]. These sensor nodes are placed in these areas to identify certain events and report back collected information to what is known as the Gateway nodes or Base Station (BS). In most cases, WSN nodes are implemented in unreachable environment where it is impossible for humans to operate. Hence it is critical for WSN to be reliable at all times in order to perform the required tasks. According to Willig [28], WSN has unique characteristics that make the design of applications and protocols challenging, the challenges which were discovered are; limited power or energy budget, restrained processing abilities, node failure, etc. Sensor nodes are the basic building blocks of a wireless sensor network, meaning that the combination of all sensor nodes constitutes the entire wireless sensor networks, and its main objective is to allow connection with other nodes in controlling and sensing the real world.

In most cases, Wireless Sensor networks architecture is designed to execute on single application or on one or more related applications [22]. There are also a few components that constituent of wireless sensor network namely: (a) sensing (b) network connectivity to all sensor nodes (c) memory to store collected information (d) computation and processing power. Sensors enable the ad-hoc network to have the capacity to monitor the environment e.g., sensors can be used to monitor disaster management, temperature, military surveillance, pressure, medical instruments and many more. Routing and transmission of packets in WSN differs from traditional wireless network, because the sensors have distinct processing activities, transmit data collected from many sensor nodes to single sink, there are energy constraints, and because nodes are deployed randomly it is impossible to ensure a unique global address, etc. Various routing protocols were proposed and implemented to cater for the

scenarios mentioned. These protocols aimed to ensure long network life time and to minimize power consumption by implementing protocols that use less energy, and can select a path in such a way that it can increase network life time. Routing protocols for WSN are categorized into four main classifications namely: Structure of the network, routing schemes, and network topology and communication models [30].

Sensors networks were initially implemented in military environments to support military applications and to observe the battlefield, nowadays sensor networking is developed for many fields because of its powerful mechanism for continually monitoring difficult locations and where it is impossible for human to operate. Current drawbacks in WSN include low battery energy. Too much usage of energy introduces problematic issues such as hardware or software failure, malicious attacks, and failure of the sensor nodes [31]. According to Sabharwal et al. [32], most WSN devices have a single radio which can receive and transmit in one frequency at certain time. In a wireless network, control plane and data plane share the same bandwidth which is available including the communication link. This can reduce the quantity of data that must transmit via a link, and actually adds the delay that can cause failure of the node. Parmar et al. [33] discovered that weaknesses of the WSN emanate from resource constraints of the sensors which include low storage and energy capacity, low computation and processing power as well as low communication bandwidth. Of these challenges, energy consumption that needs attention during the implementation of a WSN. In order to save energy, it is obligatory to deploy massive nodes which can be redundant when idle or jobless and allow their close nodes to manage their sleeping schedules so as to lengthen the node energy life-cycle and still satisfy the network observability and connectivity requirements [28]. Moreover, the links between the sensors are always prone to failure as a result of resource constraints as well as other security threats. In summary, a WSN design should accommodate fault tolerance, scalability, saving of time and memory capacity, be energy-efficient, and also rely on distributed protocols to manage any unwanted scenarios.

2.3. WSN Challenges

As highlighted in the previous section, sensor nodes can be deployed in an open and unattended environment, but they are prone to faults and attacks that can occur as a result of failure on the hardware, depletion of battery and other many reasons. This type of challenge requires a fault tolerance technique that can be considered to allow sensors to continue with their operation in the event of faults and failures [34]. WSN characteristically has power consumption

constraints. The absence of connection usually implies that there is lack of power supply to back up the sensors battery packs [35]. Although other energy gathering mechanisms already exist, these techniques typically offer an uncertain amount of operating energy. It is crucial to increase the battery life sensors node in order to ensure that the WSN is always functional [36].

a. Wireless Sensor Network Security

WSNs are vulnerable to attacks because of their nature of transmission. They have vulnerability because most of the time nodes are deployed in environments that are not protected physically. The attacks can be categorized as passive and active attacks [37].

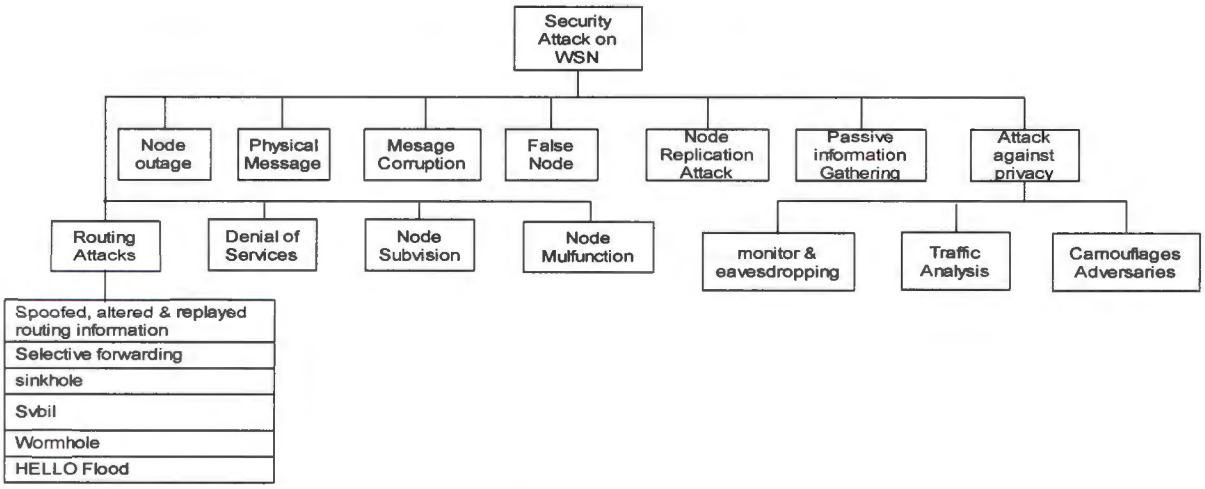


Figure 2.2: Security Attacks on WSN [37]

In Active attacks, the attackers can modify, listen and monitor data streams in the communication channel and collect information. This can introduce issues like node subversion, denial of service attacks, corruption of messages and many other issues. While on the other hand Passive attacks are known as attacks where by attackers can listen and monitor the communication channel. Examples of this type of attack are; Monitor and Eavesdropping, Camouflage Adversaries, Traffic Analysis, etc. Figure 2.2 shows security attacks on WSNs.

b. Energy Consumption (EC) [38]

Energy utilization is a major challenge in WSNs, as the lifespan of a sensor node is not infinite, so this challenge drives the design of WSN applications and protocols in several ways: A node that is always receiving or transmitting packets and continuously executing computations will run out of energy fast. Bear in mind that nowadays most laptops can operate nine to ten hours. Therefore, wireless sensor nodes hardware components must be designed in such a way that

the transceiver can switch to a sleeping mode with ultra-low energy utilization when the node is not busy, or switch it off entirely. When the sensor node is in a sleeping mode it cannot monitor the environment or partake in forwarding packets.

Fault Tolerant Aspect: To ensure that the sensor network is connected and that the environment is accurately monitored regardless of sleeping sensor nodes, it is essential that redundant sensor nodes are set up to take over in case of critical error and provide back-up in case of failure [39]. In actual fact, an important approach in WSNs is to employ extensive node redundancy and allow other nodes to observe their sleeping mode schedules in order to lengthen the sensor node lifespan while still fulfilling network monitoring and network connectivity. A couple of events are implicated, i.e, the process utilizes power, there are possibly more contestants on the channels bandwidth and routing protocols that determine efficient low energy consumption routes and have to address time variable topologies. When designing sensor nodes it is advisable to select distributed protocols and algorithms over centralized protocols and algorithms as much as possible.

c. Collision, Routing and Communication [28]

Collision is an event where an on-going transmission between the receiver and a transmitter is used by a different transmission from another station but deployed on the same channel. This event also depletes or absorbs energy of the receiver and the transmitter which results in creating extra load on the channel during packet retransmissions.

In most cases collision is caused by concealed terminals scenarios for CSMA protocols and this scenario of hidden terminals occurs when a sensor node has restricted transmission area (see Figure 2.2) consists of nodes 1, 2 and 3 organized in a way that node 1 and node 2 can interact with each other, node 2 and node 3 can communicate with each other, but on the other hand node 1 and node 3 cannot interact with each other. Assume that node 1 has an on-going transmission to node 2. Node 3 wishes to start a new transmission and senses the channel wirelessly. Because it cannot interact with node 1's transmission, then node 3 concludes that the channel is not busy and begins to transmit also. The communication between nodes 1 and node 2 overlaps at node 2 and is destructed.

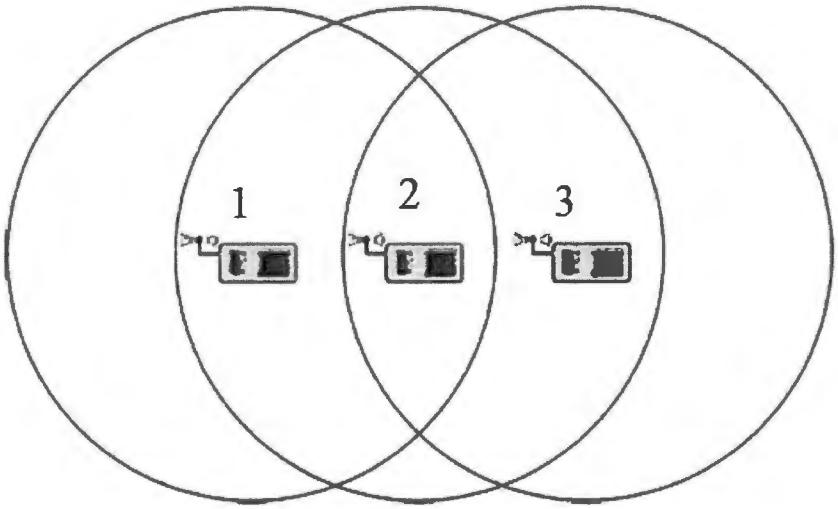


Figure 2.3: Hidden terminal [28]

There are proposed solutions dedicated to solve this problem, such as using Time-Division Multiple-Access (TDMA). WSN requires a distributed protocol to assign time slots to all nodes, however such distributed protocol tends to be very complex to implement. Furthermore, TDMA demands very tight time synchronization among nodes to prevent overlap which can result in collisions.

2.4. Overview of Software Defined Networks

Software Defined Network (SDN) is a powerful emerging networking approach which was initially introduced for wired networks, as it contains the ability to expand the traditional way wherein networks were managed and designed [40]. It is an approach to network programmability, developed to simplify the ability to manage, change, and control the network performance intelligently. It is widely adopted and advanced protocols like OpenFlow are getting improved, but as different new solutions are being introduced there are new challenges that need to be identified. In this section we are going to discuss different challenges that are faced by SDN. Inside the SDN controller resides interfaces that are programmatic. An SDN system consists of a centralized software program named a SDN controller that has a global view of the entire network such that it supervises and influences the behavior of the whole network. This framework allows the control of centralized data path elements to separate from the network technology used to communicate these devices [41]. The architecture of SDN is structured in such a way that the data planes and control planes are separated and network brainpower is centralized [11]. It has a better security mechanism that can react quickly when security threats arise, it also has powerful abilities for managing traffic, and other advantages

mentioned in [1], [2], [42]. Figure 2.3 represent a Software Defined Network architecture that consists of separate controls from forwarding hardware such as a firewall. The lines that are dashed define the control lane link while the solid lines define data plane links.

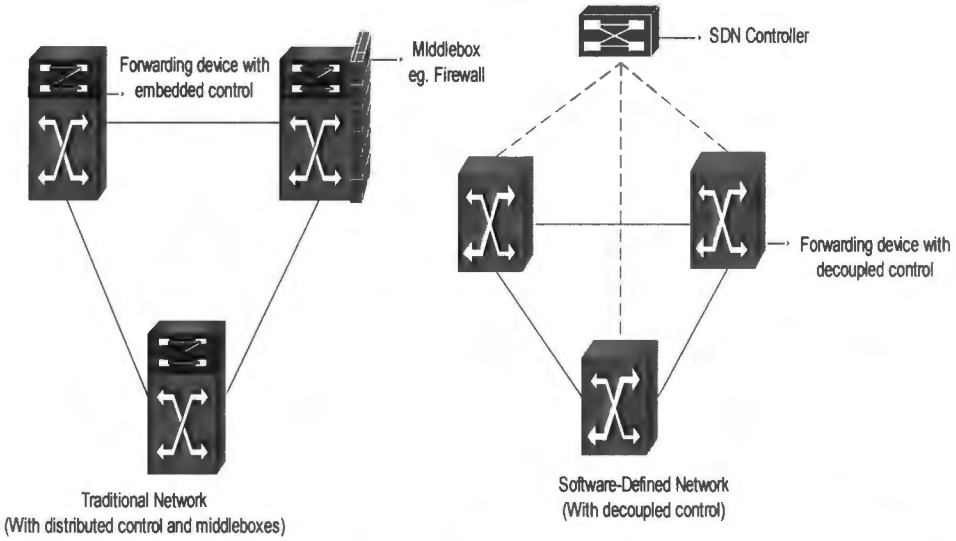


Figure 2.4: Traditional network and software defined network architecture [10]

The SDN controller makes the network to be efficient and it can also perform the CRUD (Create, read, update and delete) operation to flow entries. The OpenFlow [8] protocol enables connection between switches and the SDN controller because OpenFlow protocol has the capability to manage devices in the network. Hence, the SDN Controller oversees the entire network because of the data it gets from the OpenFlow. The OpenFlow protocol supports 3 message types namely; the controller-to-switch, asynchronous and symmetric message, each having multiple subtypes [8]. The controller-to-switch messages are initiated by the controller and are used to directly manage and inspect the state of the switch, an example of that would be a feature request message which is used to get features of the switch or flow mod message to modify the flow of the switch. Asynchronous messages are initiated by the switch and are used to update the controller about the network events and changes to the switch state. An example of this would be flow removal message because of a hard time out which means that a flow entry must be removed after a certain period of time. Symmetric message can be initiated by either the controller or the switch and can be sent without solicitation. OpenFlow forwards information to the SDN controller, giving it access to have an overall view of the network that allows the Controller to make appropriate actions. Implementing a distributed SDN controller introduces major advantages that address a lot of challenges or issues pertaining to fault tolerance, controller synchronization for consistency if one controller can fail, including many other challenges.

Additionally, the SDN controller manages network discovery and gathers information related to the network traffic by means of a special field in the flow rules earlier installed by the controller [11], [42]. On a SDN the network connectivity must be uninterrupted, it must provide absolute guarantee that the environment is detected and monitored at all time and fault can be tolerated i.e., if that node fails there should be a mechanism which will allow neighboring nodes to take over. Distributed controllers can be used to minimise latency and improve fault tolerance and scalability on SDN controller. Modern OpenFlow protocols support the deployment of more than one SDN controller that can connect to switches, the advantage of using this architecture will enable backup synchronised controllers to continue operating if failure occurs on the master controller.

2.5. Software Defined Network Benefits and Application

SDN is applied in many environments that are network related. By separating the control-planes and forwarding planes, the networks support customized management and control, creating the advantage of reducing network appliance or middle-boxes by introducing their functionality within the SDN controller, which will simplify deployment and development of emerging network protocols and services [43]. There are various environments in which SDN-solutions were proposed and implemented. Most enterprises operate a huge complex network that requires advanced security and acceptable QoS [44]. Additionally, various enterprise environments may vary in terms of characteristics, requirements, user and other factors. For example, in academic institutions networks is one of the most important with regards to enterprise networks, because most of the devices are temporary connected to the network, which must maintain bandwidth allocation and security at all times.

It is significant to maintain sufficient management in such environment, therefore SDN can be adopted to programmatically offer and introduce suitable network policies, monitor all the activities in the network and also control the performance of the network. SDN can also be used to offer integrated management and control in the network in a case where network appliance with complex functionality cannot be implemented without affecting the performance of the network [45]. Data centers have grown tremendously recently, continuously aiming to keep up with the rapid and increasing evolving and changing demands. However, one of the major demands is energy utilization [46]. Elastic Tree is a network power utilizing manager that uses SDN to locate low network-subsets that can meet the requirement of current traffic and set switches that are not active to sleeping mode [10], [37].

This technique ensures that energy is saved, which can then be used parallel to manage servers that are not used. Although this benefit is significant, not all SDN solutions can offer this benefit [47]. Managing data traffic as flows enables OpenFlow and SDN networks in particular, to integrate and support various network technologies which makes it possible to provide central control on optical transport networks and facilitate interaction between both packet and circuit-switched networks. The advantage of using SDN optical networks ensures that there is better optical transport network management and control, which provides flexibility and enables the deployment of new services by supporting virtualization [10].

2.6. SDN Architectures

SDN architecture is made up of different types of layers, as indicated in Figure 2.5. The architecture presented in Figure 2.5 is known as a trifold SDN [40]. Each layer is responsible for handling certain functions that are specific. Figure 1(b) represents the SDN layers, Figure 1(a) and (c) gives a picture of the SDN system design and plane oriented view respectively [40]. The layers are connected through several application programming interfaces. The application plane connects with the many network applications. The second layer is the control-plane, which houses the control software.

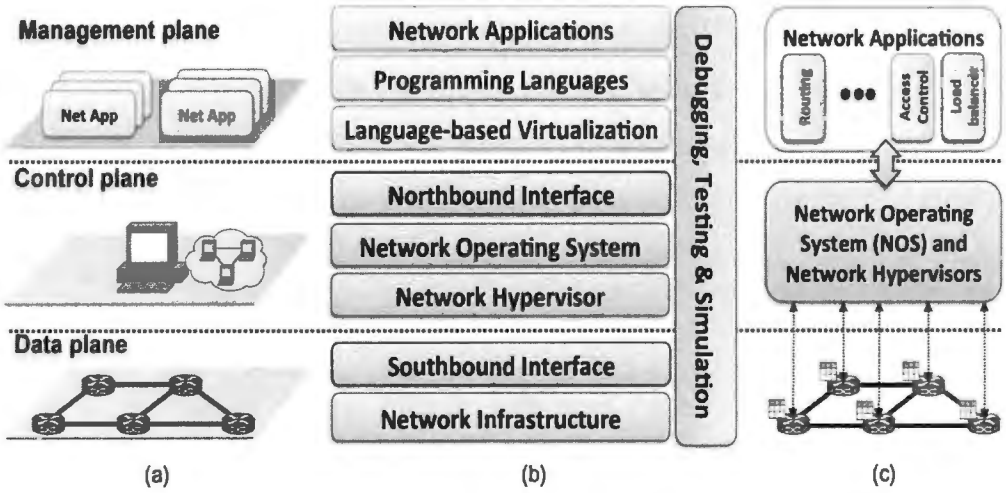


Figure 2. 5: SDN system design architecture SDN [40]

The third layer is called the data plane which comprises of network devices that are responsible for forwarding data (e.g., router, switches etc.). The interface between the application plane and the control planes is defined by application programming interfaces as denoted on the diagram as northbound interface. The communication between the data plane and the control

plane is made possible by the application programming interface called the southbound interface.

2.7. Software Defined Wireless Sensor Network

Wireless Sensor Network (WSN) is an interesting and challenging research field; it supports the evolving of applications that allow smart object such as sensors, to connect with the physical world. WSN is made up of multiple tiny sensor nodes that are resource and energy constrained, consisting of actuator and sensors that modify and monitor the state of the physical environment wirelessly. Examples where sensor network is applied is on sensing indoor, environment and traffic monitoring, military defense, disaster relief, biological applications and so on [3]. Sensor network is made up of single sensor nodes, but as the emitted energy of the single sensor node is small in most cases, Communication area is limited to a short distance which means that the network must function in a multi hop style transmitting packets over long distance. On an ad-hoc networking there are differentiated gateways, and stations connect with arbitrary node based on or subject to the station's discretion.

The situation is slightly different from on sensor networking, because frequently few sink nodes are available, to which the sensor nodes (source nodes or data sources) make their data known. Sink nodes control and configure sensor nodes operations. At times sink nodes are more effective and essential than normal sensor nodes, examples of sink nodes are desktop computers, laptops or PDA (Figure 2.6, middle figure). A sink node is a point whereby users interact with the sensor nodes by executing queries and analyzing the output. This type of a sink node acts as a normal transition point among WSN and other networks such as the building automation networks or Internet (Figure 2.6, right figure). [48].

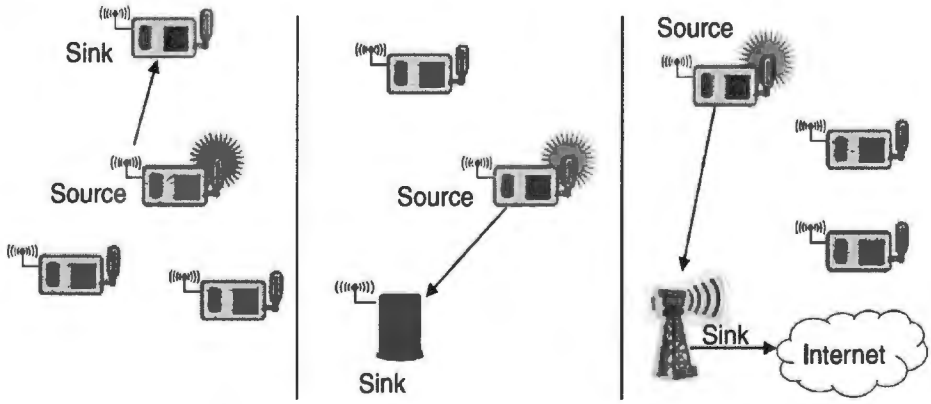


Figure 2. 6: Types of sinks in a WSN [45]

A fundamental importance of SDN in WSN is that the sensor node becomes a dump element which only understands controller messages or commands. This makes it difficult and improbable to be used as a conduit of malice. Another advantage is on the basis that sensor nodes are the peripheral devices on the network; unlike traditional networks where the host computers, which are peripheral, could also be security targets. The SDN model also enables flexible configuration moving away from the cumbersome and error prone manual process currently in place. Flauzac et al. [42] proposed a multiple controller SDNCH architecture that contains a new method for deploying a distributed security solution. Flauzac et al. [42] introduced three different components that the Domain is composed of, which includes SDN Cluster Head (SDNCH) defined as the domain coordinator; Sensor Nodes- which are defined as a group of the nodes and the gateway nodes; Gateways- which is the bridge between the SDNCH and sensor nodes. In this solution the cluster controllers interconnect SDN domains through border controllers which target on producing a WSN ad-hoc networks model which is well secured. Flauzac et al. [42] further investigated finding a solution to implement a distributed security, whereby the SDNCH can be used to control the movement of traffic among each sensor node collaboratively. The disadvantage of the introduced SDCSN is that if it gets disabled or down this will affect the entire network meaning that the whole domain will also be down. Hence in this research we want to propose a synchronized distributed SDWSN which is fault tolerant on the SDWSN controller and the sensor. This means that if one controller shuts down another controller will continue to manage the entire network. The SDNCH approach has a few advantages such as; the controller manages the network efficiently and has full control on managing the security of a single SDN domain. Huang et al. [49] presented a SDWSN prototype whose focus was to aid the power efficiency and flexibility of WSNs for environmental detection applications. The results from that study illustrated that the proposed SDWSN prototype can perfectly aid the power consumption efficiency and self-flexibility of environmental detecting in QoS. Kim et al. [50] emphasized that there is inadequate work and practice on the implementation of fault tolerance on software defined network. They [50] concluded that during the implementation of SDWSN, it is imperative to plan the architecture of the controller and put more effort in enabling the SDWSN controllers to have performance efficiency, to increase scalability, security, distribution, the ability to synchronization, etc. Making the Controllers distributed will address many challenges such as fault tolerance, synchronization, consistency, load balancing, and latency, amongst other existing challenges. De Oliveira et al. [19] presented a TinySDN solution for SDWSN that can support the implementation of multiple controllers. De Oliveira et al. [19] also studied how to

programmatically minimize data which is switched between the controller and sensor nodes. They [19] claimed that their SDWSNs solution provides more opportunities better than the normal WSNs and a prototype has been implemented. The proposed SDN Controller Node has multiple controllers which execute the same SDN controller tasks; these include the execution of certain applications, controls and creating of network flows. Figure 2.7 displays two different components which include controller server and sensor mote module.

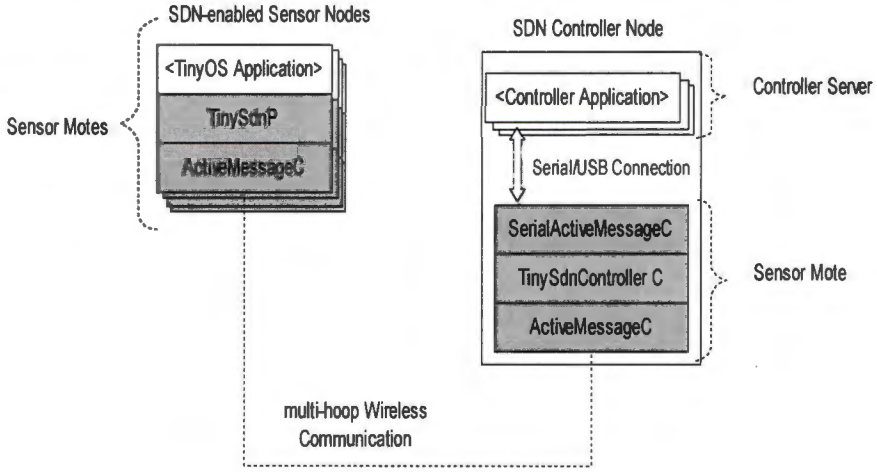


Figure 2.7: Architecture module [19]

Each SDN-enabled sensor node, where the data plane is executed, connects wirelessly with other SDN controller nodes, whereby the control planes logic gets executed, enabling communications between both planes.

Miyazaki et al. [12] also proposed a SDWSN concept and prototype system, the system can be modified by inserting roles to its Sensor Nodes (SNs) through wireless communications; moreover, these roles can be injected even after the network has been deployed. The compiler produces roles for SNs depending on the user-defined scenarios that outline the functionality that needs to be initiated in different nodes and as a result the produced roles will be delivered to suitable sensor nodes through wireless communication. The proposed SDWSN aims on injecting role programs to configure the sensor nodes, in order to simply alter the behavior of a WSN. In addition, in order to maintain the development of role programs, [12] created a scenario compiler that produces roles from a user-defined scenario narrative. In [48] Hu et al. presented a Light-weighted Edge Network Virtualization approach that aims to centralize the rational module that hosts in wireless access point per client grounded by the strategies controller's and SDN technology for implementing a central and optimal function management. Hu et al. studied the mobility management which enables the enterprise wireless local area to be more stable and flexible.

2.8. SDWSN Advantages and Applications

According to Guowei et al. [31], the messages or packets must be transmitted through a path that consumes low battery power, on SDWSN the controller has the intelligence to detect the shortest optimal path. The implementation of SDN with wireless Sensor (WS) nodes offers an improved view of the WSN paradigm [42] and introduces other opportunities as well. Because of the dissimilarities between traditional wireless networks and Wireless Sensor Networks, SDN is enhanced with extra functionality that allows efficient support on WSNs applications and protocols. Various papers highlighted this principle in depth, including Smart [9] and Software Defined Wireless Network (SDWN) [51].

Sensor OpenFlow [8] introduces a tangible SDN design allowing the support WSNs [8] and also data aggregation. It also introduces new classes responsible for forwarding rules to the OpenFlow in order to satisfy the requests of WSN and also allows energy optimization. Another objective for Sensor OpenFlow is to take into account multiple application operation and protocol evolution. The flow on SDN is address focused, but this doesn't apply to a normal WSN, as data is collected for a certain expectation or requirement. Sensor OpenFlow allows generation of flow tables through using another possible IP to influence low energy devices like Blip, uIPv6 or uIP. OpenFlow monitors the communication that occurs between the data plane and control by using the exiting TCP/IP [52]. One of the design challenges on Sensor OpenFlow is that both the data and the control traffic is used on the same network, while on SDN there is a dedicated channel which is used by the SDN Controller to interact with the data plane. Normally, it is the controller that accepts regular requests from the sensor which allows flow table entry response to listening to an incoming transmitting packet and the SDN controller will forward back the results from the response, that includes expire time related to the entry. When the WSN topology changes frequently, few or one sink manages the control traffic while a heavy request of packets can be received or sent between the sink to the sensors. In order to reduce the burst request rate due to the change of the topology, it is imperative that the sensor must not be permitted to request any flow entry to the same address, until it receives the reply or when there is timeout. SDNWISE [53] influences SDN to reduce the complexity or extent of policy implementation, to promote WSN re-configurability without depending on the vendor for minor support. In most cases sensors operate on local tasks without, and not with, controllers which are global, this is one of the reasons that makes SDNWISE a solution which is more focused on OpenFlow techniques. It can support one or more controllers by slicing, whereby a single controller functions as a proxy between the rest of the controllers and

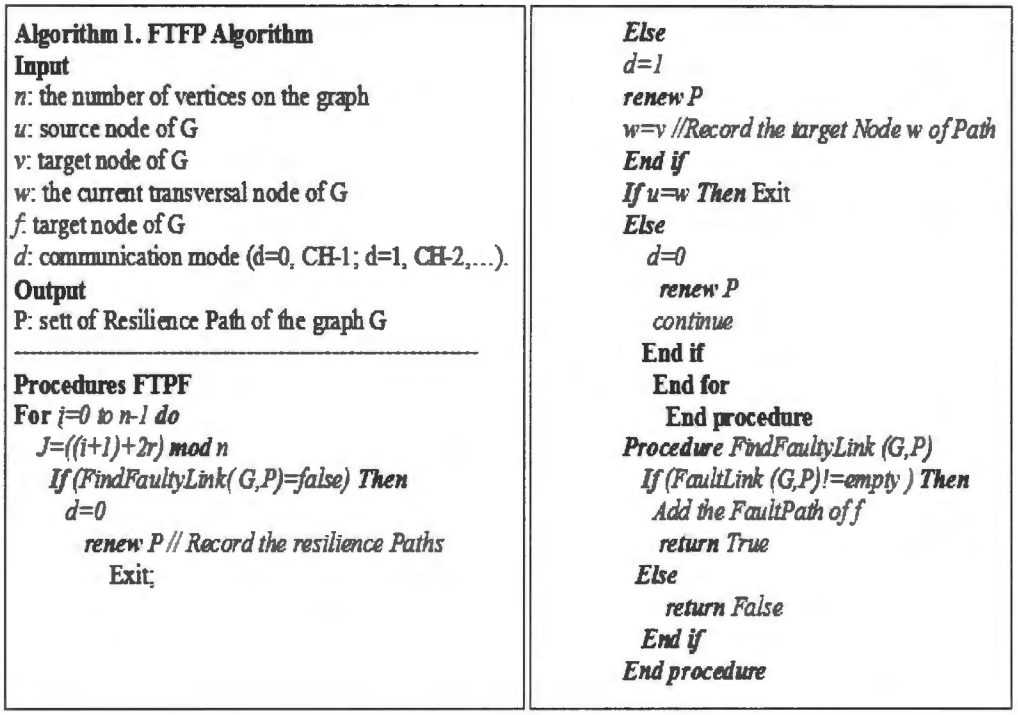
data plane. On the SDNWISE the SDN controllers and sensors are extended with the needed data structures to manage the states, to control accepted neighbor IDs and to monitor flow tables. The message from the controller to sensors or vice versa depends on the called topology discovery (TD) which is the local routing protocol acknowledged by the sensors. This learned local topology is occasionally forwarded to the main controller to enable it to have a global view network. The centralized controller forwards these messages to allow the sensors to discover the best node to the controllers.

2.9. Fault Tolerance Techniques

Fault Tolerance (FT) is a technique that gives the system the ability to keep on performing its task in the event of faults or errors [54]. Since the controller is responsible for handling all decisions on the network, it detects issues regarding fault tolerance on the network. SDWSN is deployed by well-known companies such as HP, Google and many more because of it flexibility. Even though SDN is introduced to handle issues that traditional WSN cannot handle, there are still major issues that needs to be handled: network failures and FT. If there are no proper FT techniques, network device failures can block normal packets forwarding even though there are other alternative paths in the network that exist. So it is critical to ensure that the network is reliable and fault tolerance. There is still insufficient experience and research that focuses on handling fault tolerant on SDWSN networks. In our research we aim to provide an efficient FT mechanism which will assist the SDWSN controller and the sensors to recover from various failures emerging from any fault domain on SDWSN network. On controller based faults we intend to investigate Fault Tolerance techniques that can be adopted to handle FT on the controller.

Kim et al. [50] proposed CORONET, a FT technique for SDN. In their work they claimed that their fault tolerant mechanism support path in the network. CORONET is focused on quick recovery from link failures in a few seconds after detecting that a fault has occurred. CORONET allows scalability to large networks which means that when the size network increases the flow table entries are still manageable, supports data forwarding rate with minimum traffic control and fast failure detection. Another advantage of CORONET is that it supports any type of topology, for example, Mesh, Fat tree, clique topology, etc. The disadvantage of this technique is that it only supports single controller architecture. This technique is not suitable for us because we are focusing on a distributed SDN controller with

multiple controllers. Kim et al. [50] also indicated that CORONET supports on a single controller which allows it to have a global view better than distributed algorithms. For tasks that can be done locally CORONET depends on the switch and routing technique, it does not need to use the global view. The disadvantage of this architecture is that if a single fault occurs to the controller, the entire network will be off line. Chen et al. proposed a Fault Tolerance Path Finding FTPF [55], Figure 2.7 presents an algorithm for FT path finding algorithm.



Algorithm 1. FTFP Algorithm
Input
n: the number of vertices on the graph
u: source node of G
v: target node of G
w: the current transversal node of G
f: target node of G
d: communication mode (d=0, CH-1; d=1, CH-2,...).
Output
P: sett of Resilience Path of the graph G

Procedures FTPF
For *i=0 to n-1 do*
J=((i+1)+2r) mod n
If (FindFaultyLink(G,P)=false) Then
d=0
renew P // Record the resilience Paths
Exit;
Else
d=1
renew P
w=v //Record the target Node w of Path
End if
If u=w Then Exit
Else
d=0
renew P
continue
End if
End for
End procedure
Procedure FindFaultyLink (G,P)
If (FaultLink (G,P)!=empty) Then
Add the FaultPath of f
return True
Else
return False
End if
End procedure

Figure 2. 8: Fault tolerance path finding algorithm [55]

Desai and Nandagopal [56], investigated how to handle link failure on SDN networks, but their system did not fully recover from all faults encountered. In this paper we are going to focus on two fault domains in SDWSN: (i) Controller, where the controller machine fails and (ii) Sensor, where the sensor fails. There are various reasons that make it difficult to design a proper fault tolerant system that can be implemented on the SDWSN. For example, failing to do a proper extensive modification and testing when adopting existing FT systems that normally operate on a traditional network which is not fully supported by SDWSN Controller application. Longo et al. [57] suggested that to avoid single point of failure on the controller, proper redundancy policy ought to be implemented, and also focusing on eventual overheads due to the network management, data consistency, and similar problems.

2.10. Mobile Agent Fault Tolerance

Mobile Agent (MA) is one of the fastest developing and evolving technologies [58]. A mobile agent is an autonomous piece of code that has the capability to perform certain tasks on behalf of a system or a user [59]. MA is a programming paradigm that is supported on many distributed applications which is different from other computing paradigms because the code that acts on the data gets transmitted among nodes [60], [61]. Application development is made flexible due to the transmission of the code among nodes. MA is very flexible such that it can defer the execution, travel to another node and resume the pending execution on the new node [62]. An agent can communicate with other agents and make use of their resources when executing specific tasks but maintaining their self-independency [63]. Using mobile agent introduces a lot of benefits on computing which makes them very significant on distributed environment.

For an agent to remain reliable and available it must not fail when a node fails. The same should apply to a node, i.e., a node must not fail as a result of agent failure. This means that the agent should not depend on the node and vice versa. Usually agents fail when a host encounters failure or in a case where an agent might arrive at the desired destination. MA can be implemented in distributed network environments and perform their computational function when migrating from one host to another. There are a lot of challenges concerning reliability of agents. It is important that the agent must be reliable. Therefore, fault tolerance for MA and MA systems is considered to be significant. An agent must offer FT in order to deliver an efficient resistance mechanism against failures and provide required results. In [60] Yousuf et al. highlighted that, an agent is one of the powerful mechanisms that can be used to provide fault tolerance in a system. Extensive work has been done on areas such as fault tolerance and survivability using MA mechanism [25], [64], [65]. Therefore, its technique can be used to offer fault tolerance on the controller, nevertheless we need to understand its benefits and challenges.

2.11. MAFT Challenges

Computing system faults can occur due to various reasons which include program bugs, unforeseen states, processor faults, and communication fault, leading to unwanted behavior [66]. Pleisch et al. [67] proposed a MAFT which is centered on two roles namely; sniffer and correctors. The sniffers monitor the activities of the network at various stages, to perform a certain task it has the ability to travel from one node to another to check faults and then perform

alternative solutions. It monitors the place a where fault exists and informs the correctors. Correctors are responsible for restoring agents at various phases on the network. Software and hardware components can be subject to failure in a computing environment, failure can be of an agent, a machine and a place. Other conditions are presented in details in Figure 2.9:

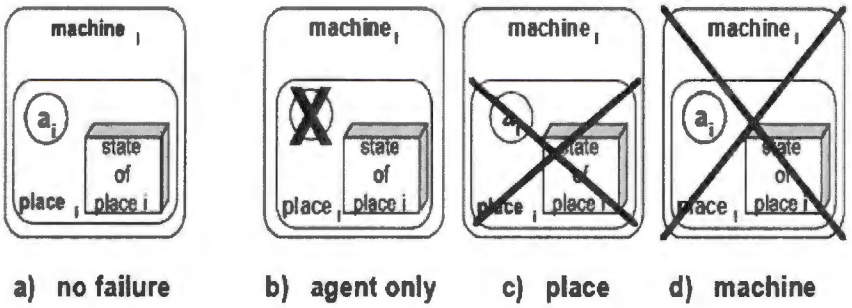


Figure 2. 9: Failure model for mobile agents [67]

In [60], Yousuf et al. outlined different kinds of faults that can happen in agent's environment, which includes: Node failure, Agent failure, Fault of component of the agent system, Network failure, Loss of message etc. Yousuf et al. outlined that it is not simple to define unwanted states of a Mobile Agent due to the fact that it adjustive and heterogeneous in nature, meaning that since it is independent there is no global control. In this study we want to implement a FTMA that has a global view of the internet in order to monitor fault.

A lot of solutions which manage the problems outlined above have been proposed. Our aim is to find a mobile agent which can be used to manage fault tolerance on the controller. SDWSN contains powerful mechanisms that can introduce effective communications technologies in the future, but there are still critical problems that need to be resolved in order to reach the required goals. SDWSN is undergoing serious development and improvement, however little work has been done where the focus is to use MAFT to manage FT both on the SDWSN controller [25]. Although a lot of research has been done to address SDWSN challenges; a lot of challenges still exist such as node failure, FT on the controller and the sensor node, controller scalability, synchronized distributed controller etc.

Another challenge that still exists is managing security on the controller, because if it the security SDN controller has been compromised, the hacker will attempt to shut down the network. In summary, we need a fault tolerance technique that can ensure fault tolerance in the event of failure on the node and the controller. This will increase network dependability, and fault tolerance on the controller.

2.12. Related works on SDWSN and controller Fault Tolerance

In SDWSN, the controller plays a significant role of enabling intelligence of the whole network.

Its focuses on network programmability, mapping function and flow rule generation [17]. The model for software defined network enables additional functionality. When designing the SDN, the control plane can be implemented in different ways, for example the controller can be centralized [68], clustered or distributed [69], [70]. Centralized controllers are considered in [71]. Figure 2.10 and 2.11 presents a distributed controllers and a centralized controller.

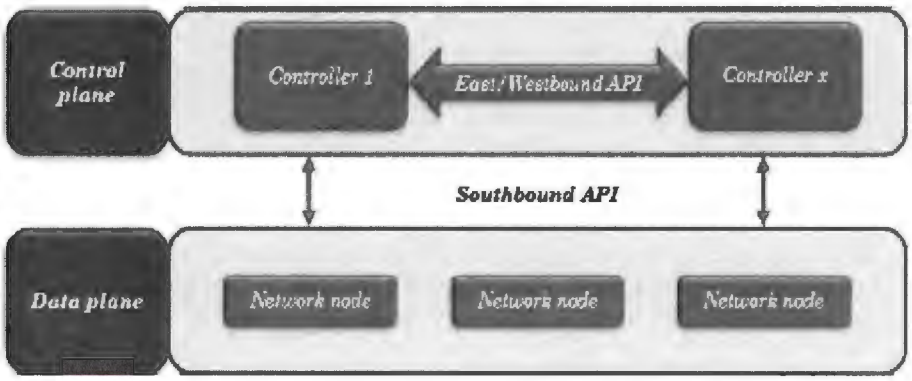


Figure 2. 10: Distributed controller [17]

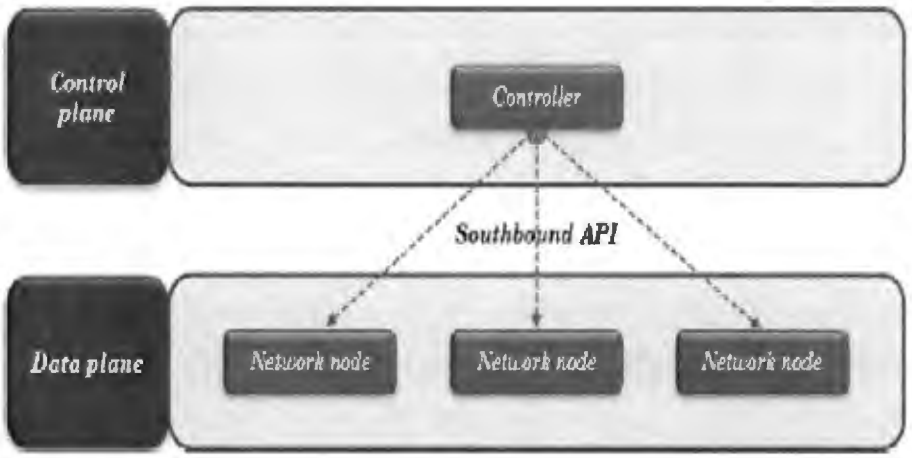


Figure 2. 11: Centralized controller [17]

Authors in [72] [73] [74] proposed solutions that can dynamically reduce or add controllers depending on network load. Costanzo et al. [51] proposed a Software defined network architecture that consists of embedded controllers that are connected to the sink node via a serial-connection. Communication between the sink node and other sensor nodes is achieved wirelessly through wireless interface. A virtualizer, adaptation module and the controller are incorporated in the embedded system running on the Linux. The virtualizer enables the collected information to be arranged to form a virtualisation of the network state to the controller [51]. The disadvantage of this technique is that the serial-connection between the controller and sink node is limited, which affects the interconnectivity of the sink node and the controller to have one-to-one relationship. It also introduces the issue of scalability. On the other hand, message formatting is achieved through the adaptation module. In [75] [9], on the architecture of the SDN the BS acts the same as the Sink which can also support the controller. The SDWSN that was proposed by Gante et al. [9] is made up of 5 layers namely; Application, Medium Access, Physical, NOS and Middleware. The middleware is made up of the mapping function, mapping information, definition of flow tables and the controller. The controller is responsible for controlling and managing the entire network and ensures flexible network programmability.

In order to manage and monitor the network, the controller uses the monitoring messages to gather information on the network, the information that includes the energy levels of the sensors, fault detection, etc. Mapping function is responsible for processing the data collected from the sensors and the Mapping Information module then stores the information acquired. Application layer focuses on locating areas that need to be sensed, this implies that the application layer has the tracking and location algorithms that ensure the maintenance of accurate information on the position of all sensor nodes. A cluster-based SDWSN that has base station was proposed by Olivier et al. [8]. One advantage of this approach is that the software defined cluster sensor network SDNCH is able to implement policies that can ensure secure domain and has partial view of the network.

Huang et al. [76] proposed a prototype that Energy-Efficient Monitoring in SDWSN Using Reinforcement Learning to help with handling load balancing and redundancy and to solve challenges of energy consumption of WSN environment [76]. TinySDN [70] supports multiple controllers. TinySDN was proposed by Berde [70], the controller is serially connected to the sink node and sensor node. Cooja simulator [77] was used to test and implement TinySDN. The Collection Tree Protocol (CTP) establishes the connection between SDN sensor node and

the controller. Information such as link quality is also sent to the controller using the CTP protocol. Jacobsson et al. [78] proposed a hybrid control architecture that has a local controller on each sensor node and on the main controller on the control-plane. The local controller is responsible to monitor, configure, setup, reconfigure, and execute commands that are directed from the controller. New code patches or new protocols can be installed on the virtual machine prepared for the sensor node.

2.13. SDWSN and Traditional WSN Comparison

The implementation of a SDWSN demands the integration of a large number of hardware and software components as. It is made up of distributed sensor whereby the data that has been collected can either be processed locally or transmitted to a remote site through a broadband network. This type of network can be implemented to support agriculture applications whereby several environmental factors are detected over a large environment. Table 2.1 below represents related work on SDWSN Energy Consumption as well as on traditional WSNs Energy Consumption. Table 2.1 shows the difference between SDWSNs and traditional WSNs.

Table 2. 1: SDWSN and traditional Wireless Sensor Networks [18]

Type	H[a]	D[b]	Objective	Solution
SDWSNs	✗	✗	Reduce rigidity to policy changes and management issues	SDWSN architecture is proposed and address the main technical challenges
	✗	✗	To reduce the amount of information exchange between Sensors and controller, and to program sensor nodes as finite state machine	A prototype of SDWSN is implemented
	✗	✗	To reduce the amount of control data in SDWSNs	TinySDN is designed and implemented that enables multiple controllers in SDWSNs
	✗	✗	Energy minimization in multi-task SDWSNs	An energy-efficient sensor activation strategy is proposed
Traditional WSNs	✓	✓	Optimize the energy charging by sensors while minimizing disruption to data communications	RF-MAC protocol is proposed
	✓	✓	Optimization of energy delivery to energy hungry sensors	MAC protocol is proposed
	✓	✓	Maximize the number of nodes receiving power	Deploy mission aware placement of wireless energy transmitters
	✓	✗	Optimize the design of energy storage capacity of sensor nodes and minimize the average packet delay	An optimal design is proposed which consider the delay sensitive and insensitive scenarios
	✓	✗	Balance the available energy among sensor nodes	A cross layer approach is used which considers routing, scheduling, and power control
	✓	✓	Overcome the path losses of RF signals and to enable sensor nodes to charge energy from a distance	A directional energy transmission/ reception model is proposed
	✓	✓	Placement of minimum number of energy transmitters	Energy transmitters are placed while jointly maximizing total energy and fairness
	✓	✓	Joint optimization of energy transmitter placement and power allocation	Optimal placement of energy transmitter is done in order to maximize the charging quality
	✓	✓	Minimize the network deployment cost with minimum number of energy transmitters	Energy transmitters are placed while satisfying the harvesting and performance requirements
	✓	✓	Placement of energy transmitters	Energy transmitters are placed optimally using greedy algorithms

[a](H)arvesting or wireless power transfer.
[b](D)edicated energy transmitters.

WSNs that sense large areas need enough energy supply and the sensors are designed as XBee module, microcontroller and combination of sensors. Essentially the XBee module is divided into several devices namely; end device, router and coordinator. Coordinator has the ability to monitor the whole network while the router basically relays messages in mesh or tree topologies. The end device communicates with both the router and coordinator or just the router. Only one coordinator can exist in a network, but the number of router or end devices is unlimited. This section intends to differentiate efficiency of traditional WSN and SDWSN when using MTE routing protocols. The MTE routing protocol focuses more on reducing energy consumption during the transmission of data. The technique to maneuver this principle is easy; sensors are designed in such a way that communication only occurs between the nodes that are close to each other as data is transmitted to the gateway.

Tomovic et al. [79] conducted a study that shows the energy efficiency of a traditional WSN and SDWSN using a MATLAB simulator. The results showed that SDN lengthens the battery life of the network as compared to the traditional WSN. From the study, Tomovic et al. [79] concluded the SDWSN had more advantages than traditional WSN. Figure 2.12 shows that the SDN lifetime is last longer than the MTE lifetime.

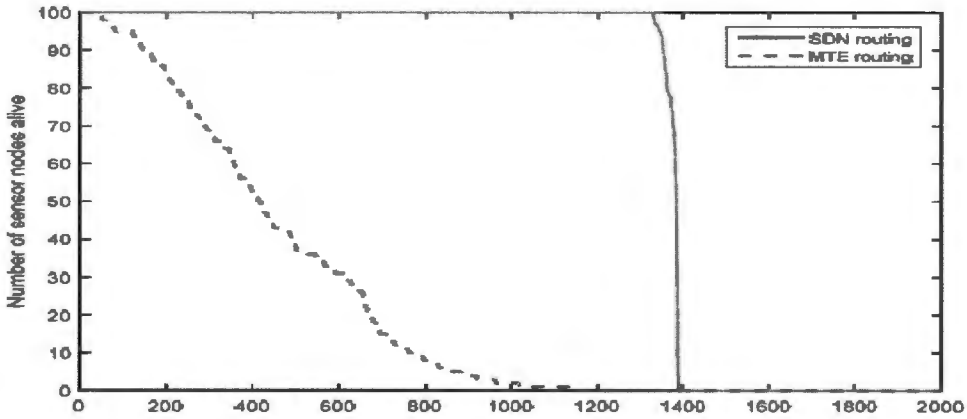


Figure 2. 12: Simulation scenario for a SDWSN and MTE WSN lifetime [79]

SDWSN is one of the challenging fields in networking [17]. It is literally the integration of two separated unfledged models [17], it is composed of an SDN model and a WSN model that are interconnected despite their complexities. A wireless sensor networks is always faced with the challenge of being resource constrained, which makes it critical to always design a network in such a way that it can support minimum energy consumption. Regardless of the efforts that were made in the past of introducing efficient techniques to cater for energy consumption challenges, there is still work to be done to fully optimize energy efficiency. Merging SDN in

WSN introduced benefits of solving other resource challenges that were in WSN, but it is still important to note that the SDN model also introduces its challenges, such as fault tolerance on the controller or constancy on distributed controllers.

2.14. Chapter Summary

In this chapter an overview of the WSN was discussed together with its application, benefits and challenges. An introduction to SDN was presented, highlighting its components, application, challenges and benefits. Later we introduced the concept of integrating SDN and WSN and outlined the challenge and benefits of using SWN to cater for challenges that were discovered. Fault tolerance on the controller was one of the challenges on SDWSN, and literature on existing MAFT techniques was discussed and explored. This chapter also compared the difference between a traditional WSN and a SDWSN. The next chapter will focus on analyzing existing fault tolerance approaches and mechanisms.

CHAPTER 3

Analysis of Fault Tolerance Approaches and Mechanisms

3.1 Chapter Outline

This chapter discusses an in-depth comparative analysis on existing FT strategies and mechanisms that can be used to handle FT on a network system. The goal is to identify a FT technique which is resourceful, reliable and has an effective mechanism to support FT on the SDWSN controller. It begins with an overview, followed by the methodology used, the parameters which will be used to evaluate suitable FT techniques and the actual analysis of FT techniques proposed in the literature in terms of fault detection and recovery.

3.2 Overview

Fault Tolerance is a mechanism that enables the network system to remain operational in the event of faults on the system. Implementing such FT mechanism increases the chances of the network remaining operational in case of failure as compared to network remaining operational in case of fault or minor errors as compared to simply designing network systems that have no proper fault tolerance and can totally break in the event of small failure. FT is required particularly in the environment such SDWSN as they require high availability of the system in order to keep the system working. Fault tolerance has been one of the significant research topics in computer science and networking. Most researchers highlighted that failure can occur in many different ways. Failure can arise when a node fails to take an action which was supposed to be taken or when a sensor node takes an action which was not supposed to be taken. Failure can also occur because there is no mechanism or technique that was configured to accommodate fault tolerance in case failure occurs. Concrete research has been made on the subject of fault tolerance [1], [80], [81]. This chapter focuses on exploring different kinds of existing FT techniques that use mobile agents and evaluate their merit on the optimization to handle fault tolerance on a network system and apply the same technique to handle fault tolerance on SDWSN sensors and controllers.

3.3 Methodology

There are many methodologies that can be used to perform the analysis of the existing MAFT. A qualitative research method was used as stated in chapter one, literature survey was also deployed as a technique to collect data. We used several research articles only explain the

proposed solution in-depth and were selected based on criteria such as FT approaches used (i.e., replication, check-pointing, or hybrid) and exception handling. Moreover, research on MAFT techniques and agent reliability were analyzed to discover how the existing MA systems work and understand the platform of applications that are built on them [82]. The focus in this case is on identifying FT approaches used such as replication, checkpointing, or a hybrid (both replication and checkpointing) as well as exception handling. The following criteria were used to select a suitable MAFT technique to be adopted: ability to support multiple distributed controllers and scalability. With scalability, it means when the size of the network increases, the flow table entries are still manageable, energy efficient, support all topologies and there is fast failure detection.

In this analysis, fourteen (14) papers on MAFT were evaluated and in-depth analyze of the techniques used were carried out. The knowledge gained will be utilized to design a resourceful FT system for SDWSN controller. Moreover, the lesson learned from the analysis is that, MAFT strategy poses a powerful FT approach that can be used to enhance network dynamics such as performance system, network computation and communication. Accordingly, in [25] it is stated that MA have already been used to handle data diffusion in WSN as well as fault detection and recovery.

3.4 Evaluation Criteria

This section discusses the criteria and parameters used to evaluate appropriate MAFT approach suitable to handle FT in SDWSN controller. The summary of the techniques analyzed is presented in Table 3.2. where, YES or NO is used to indicate whether each parameter or requirement is handled in each technique evaluated. The analysis will be based on the following parameters:

A. Fault tolerance technique

Different kinds of faults can transpire on the network system such as network failure, agent failure, lost message, node failure, and many others. This parameter will identify the type of errors and faults which is handled by the proposed FT technique, it also indicates the type of FT technique or strategy used to manage the system during the event of faults and errors.

B. Fault identification and detection

Fault identification is a process of identifying the faults that can occur on the network. This parameter shows whether or not the FT technique offers fault detection solutions and if they

exist, what type of mechanism is used. The detector will trigger a fault detection mechanism each time a fault is detected on the network or on the system.

C. Class of fault identified and fault type

Faults can occur in a network based on the following fault types: node, packet loss, server, network etc. This parameter will give an understanding of what type of fault is covered by the proposed FT mechanism. Knowing the type of fault that is handled is critical for its adoption for implementation. This is important to identify the effectiveness of the analyzed techniques.

D. Recovery mechanisms

Recovery mechanisms as a parameter will help us identify the technique responsible for recovery. For instance, the parameter will indicate whether it uses forward, or backward recovery mechanism as well as assisting us in measuring the level of FT and effectiveness of the technique.

E. Exception handling mechanisms

This parameter will help identify if the technique employs exception to allow the system to remain fault tolerant in the event of faults and errors. In Table 3.1, YES or NO is used to indicate presence of exception handling mechanims in the FT approaches.

F. Synchronization technique

This parameter indicates how the proposed technique synchronizes its contents with other agents as a guard against failure. The parameter is based on C and D above.

G. Security

Security application on the network or any system improves the efficiency of the network. Thus, security offered by the FT technique needs serious attention. The parameter is significant to check if each FT technique considered in this analysis offers security measures to ensure sustainability and availability of the agents. In Table 3.1, YES or NO is also used to indicate presence or absence of security.

H. Performance analysis

Performance analysis will show if there are any performance measures stated in each study considered.

I. Energy consumption.

Since sensor nodes are prone to error due to power failure or battery failure, the energy consumption parameter will identify if the technique offers a solution for energy consumption or not.

J. Blocking and Unblocking

This parameter will be used to show if agents are blocked or unblocked in the event of failure during execution. This is important for systems that require high availability, as it is critical to ensure that the FT technique won't make the system out of service, and it is important for implementing the FT approach.

K. Checkpointing

This parameter is used to show whether the FT technique takes a snapshot of the process state to be saved in a stable storage or not.

3.5 Analysis of Mobile Agent Fault Tolerance Techniques

This section presents the analysis and the evaluation of existing MAFT approaches focusing on parameters that were defined in section 3.4. The analysis is summarized in Table 3.2 and 3.3 in terms of advantages and disadvantages respectively.

A. FATOMAS

Pleisch *et al.* [80] proposed a java based MAFT system called FATOMAS, which focuses on crash failures. FATOMAS introduces two FT approaches namely; place dependent and agent dependent approaches. Place dependent approach provides support and integrates fault tolerance into the mobile agent platform. This FT technique implements the use of masking and replication, using replicated servers in order to mask failures. The agent place dependent technique introduced on the FATAMOS is responsible for allowing fault tolerance agent to execute parallel a protocol that enables fault tolerance. Agent dependent approach is significant because it accepts fault tolerant agent to execute without redefining the mobile agent platform. Failure on place, communication link, agent, machine, or a node can cause blocking of mobile agent execution. Figure 3.1 shows the FATAMOS architecture.

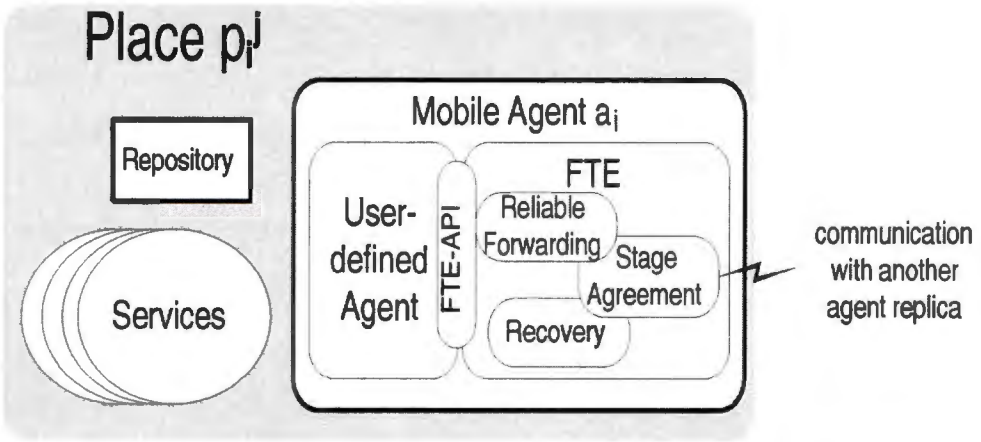


Figure 3.1: FATOMAS Architecture [80]

FATAMOS emphasizes two significant FTMA execution properties which are; (i) exactly once and (ii) non-blocking. Non-blocking enables an agent to operate regardless of failure of place, communication link, agent, or machine. Blocking is addressed by using replication. However, the disadvantage is that replication can enable multiple agent executions which violate the exactly once property. FATAMOS has two building blocks which are Reliable Broadcast and DIV consensus. Because of lack of complete transparency to the user-defined agent, FATOMAS uses Fault Tolerance Enabler (FTE) to help interact with user-defined agent to address stage agreement problems, issues on replicated iso-places and hetero-places. To provide FT, each user defined agent and the logger agent offers logging, and checkpointing. Both the user defined agent and its logger agent are responsible for monitoring each other and can create agent repair when a fault is detected.

B. Norwegian Army Protocol

Johansen et al. [81] proposed Norwegian Army Protocol (NAP), a Python based approach to Mobile Agent (MA) FT execution. This technique assumes a fail-stop model, which corresponds to a perfect failure detector. NAP is integrated into the existing architecture of TACOMA. In this FT technique, replication is achieved by leaving a computer Program executing on landing pad that the MA had visited. This technique is also based on linear broadcast strategy, no additional processors are required and no programmer defines recovery is required during MA failure recovery. In this approach the MA has three operations namely: *(i) Move, (ii) Spawn and (iii) Checkpoint.* MA can migrate to another host through the move operation, alternatively it can continue running on the same host and generate a new MA on a different host through spawn operation. Checkpoint operates similarly to move, but the new action head code runs on the current landing pad instead of the host. Checkpoint refers to information of state that was saved after execution [83]. Figure 3.2 presents the Mobile Agent

Versions. NAP consists of 5 FT activities which are: visit, launch, alert, update, report. A drawback of this technique is that if a MA terminates the NAP will also terminate, the other disadvantage is that these techniques cannot be implemented on systems that can sufferer partitions.

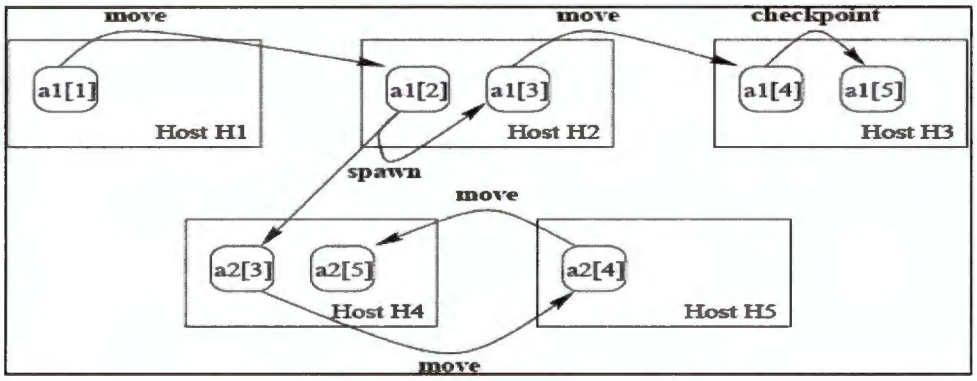


Figure 3.2: Mobile agent versions [81]

This model was supported by Schneider [64]; the protocols are based on the idea that there are multiple replicas of a place existing in the system. Deterministic results on this approach are produced as copies of the agents are executed. This approach doesn't cater for malicious and Byzantine faults. Byzantine failure handling technique is focused on the execution of multiple instances of the agent, therefore execution is not an issue and consequently not forced by the protocol. Although this approach can be implemented at low cost, its drawback is that it cannot support or handle partitions in the network.

C. Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents

Rothermel et al. [84] proposed a transactions-based solution that focuses on advancing failure detector approach, and ensures that Mobile Agent (MA) performance is exactly once property and also performs on exactly one node on each individual stage. This technique uses a voting protocol which is based on the FT majority quorum algorithm. Rothermel et al. [84] also showed how Exactly Once Property (EOP) is achieved in MA systems; they used the stage concepts to minimize the probability of MA being blocked as a result of failure. Their protocol ensures EOP and dynamically selects the degree of FT on the individual steps of MA execution. The protocol is implementable on conversional Transaction Processing (TP) monitors that are conversional and transactions-based massages. This approach has a message complexity and blocking probability, which is defined as the probability that a MA is blocked on a stage as a result of failure. The availability of stage A_s is given by $A_s(n,p)$ where n denotes the number of nodes on A and p is available individual node. A formula to calculate $B_r(n,p)$ Relative Blocking

NWU LIBRARY

Probability (RBP) is defined in detail [84]. Reducing the blocking probability makes the message overhead to be high, and when n is three or above the RBP is also reduced. The drawback of this technique is that modelling this FT is difficult and very slow [84]. This technique uses passive replication mechanism with the intention of providing non-blocking FT on agent based application. If it happens that the worker agent fails, one of the observers will be selected to be a new worker. The disadvantage of this approach is that, if the data manager gets suspended, the transaction can crash even though all managers are correct i.e., a weak Atomic Commitment does not require a perfect failure detector [85]. These techniques have high bandwidth consumption when sending the result and during the process of commit.

D. MOCA

Damasceno et al. [86] have proposed an approach which is recommended for fault tolerance by [60]. This technique uses forward recovery and backwards recovery to handle agent failure. It has the capability to enable flexibility during code mobility and physical mobility. MoCA can operate even in the event of agent failure, using the backward recovery technique. MoCA emphasizes the development of effective mobile agent applications that integrates explicit services with the objective of supporting software agents. It supports error handling mechanisms, like any software components MoCA supports exceptional and normal activities that enable the Agent to have normal activities. On the other hand, exceptional activities offer mechanism that enables the agent to operate with exceptions.

The exceptions are divided into two, (i) predefined exceptions – are associated with faults detected by the hardware, middleware or runtime support. Predefined exceptions are declared indirectly. (ii) User defined exceptions – these types of exceptions are detected from an application level. In case of fault, the device will throw an exception that notifies other mobile users about the problem. MoCA offers programmers a set of abstraction and provides FT through exception handling; its disadvantage is that it handles FT on an application level. Even though MoCA supports various techniques for handling fault tolerance in the network, their techniques might not always be applicable to other challenges.

E. Antecedence Graph Approach to Checkpointing for Fault Tolerance in Mobile Agent Systems

Singh et al. [87] proposed a parallel check pointing technique capable of providing fault tolerance in mobile agent systems. Parallel check implies that inter-agent synchronously can make use of checkpoint whereby the check pointed antecedence graphs are stored and used for fault recovery and FT support. The main objective of this approach [87] is to reduce the global checkpointing latency and to minimize the total recovery time. This approach focuses on storage ability and computation during MA execution and during transmission it minimises synchronization message. It reduces the latency during global checkpointing and minimizes the recovery period. Although this techniques (antecedence graphs plus message logs) was presented in [88] to handle FT on MA systems, it struggles from overheads because of the size of FT information and the messages when the agent increases [87]. In this technique, checkpointing algorithm helps to optimize bandwidth allocation and consumption. The proposed checkpoint algorithm is synchronized and allows a defined number of agents executing process for taking checkpoint, if failure occurs, the antecedence graphs and message logs are regenerated for recovery and then operation continues normally. The checkpointed antecedence graphs and message logs are saved in the stable storage and can be used for FT and fault recovery. It consist of Multiagent Group (MAG) which has a base agent that monitors agent, the MAG is a fault recovery manager. This approach claims to improve recovery times; minimize message overhead and execution time.

F. Chameleon

Bagchi *et al.* [89] presented an approach to fault tolerance using MA. This technique makes use of specialized agents and backward recovery to remains operational during the event of system failure and agent failure. It uses centralized management and dependency analysis is not needed. It makes use of indirect communication between the applications and agents due to daemon agents. If blocking occurs during agent execution no harm is done on the system. The advantage of this approach is that during run-time agents can be removed or added. It consists of three categories of Agents namely; (i) Manager Agents, (ii) Daemons Agents and (iii) Common agents, details of how each Agent operates is found in [89]. In this technique if the nodes fail during agent execution blocking will occur but it will not affect the system. Chameleon is scalable, accommodates other fault tolerance techniques, and can also be ported with present reliable atomic multicast. It makes use of redundant network to continue operational in the event of switch failures, link failures, or application failure. The drawback

of this approach is that, it doesn't differentiate failures between the network and the node, i.e., switch failure, or link failure. Another drawback of Chameleon is that there is extreme blocking if errors occur during nodes execution.

G. Novel Dynamic Shadow Approach for Fault Tolerance in Mobile Agent Systems

Hans [90] proposed a MAFT approach that focuses on handling crashes by creating a copy of the MA which can be used if the Original Agent fails. When the original agent migrates to another server the clone agent remains in the current server. The clone agent gets deleted when the original agent migrates again to another server creating another clone on the new server. This process is repeated until the completion. In cases where the original agent stops execution as a result of errors on the server, it can be recovered from the server. If an error occurs on both the server containing the original and the clone agent, a message is sent to the main server requesting agent replications. This is a way to enforce fault tolerance. If the clone continues to have errors, checkpointing technique is introduced and the total trip time is used to evaluate improved performance. Since data from the prior server is saved on the main server agent roll back is not required. This approach uses checkpointing to prevent the agent to roll back if it happens that the clone of the agent and the original agent fail. The objective of this approach is to reduce rollback by putting checkpoints. In this study different experiments were presented by the authors; overheads calculation, check both the clone and the original agent's total trip time with checkpoints and the impact when both the original and clone agent gets lost because of fault. The approach offers significant advanced performance and reliability. Its drawback is that it is not appropriate for applications that are time sensitive.

H. Checkpointing Using Mobile Agents for Mobile Computing System

Chowdhury *et al.* [91] proposed techniques that provide a checkpointing protocol that offers improved reliability on distributed application and enables mobile agent management. The authors introduced an algorithm for rollback recovery with a technique responsible for managing checkpoints and message logs [91]. This approach uses independent checkpointing, so messages synchronization is not required. Bandwidth consumption is reduced and log file message size is reduced and as a result this approach is regarded as scalable. The recovery time is influenced by the rate-of-failure of mobility, mobile host and bandwidth availability. When MA must recover it never goes beyond the threshold value. Mobile Service Stations (MSS) enable Mobile Hosts (MH) to communicate with the entire system through the MSS they connected to; this can be regarded as the Home Agent (HA) of the respective MH. When MH

migrates to another station, connections to the old MSS are replaced with a new allocated MSS. This approach consists of two agents, the *Log Agent* (LGA) which is located on every MSS that monitors the MHs messages logs and the *Checkpointing Agent* (CPA) which is located at the HA. If the MH is recovering after failure occurred, Checkpoint Transfer Request (CTR) is forwarded to the MSS that has recent checkpointing details and the Log Transfer Request (LTR) is then sent to the rest of the MSSs. From this technique the total recovery cost and the total checkpointing transfer time (CTT) can be calculated, and the results of the existing checkpointing techniques are compared.

I. Using Host Criticalities for Fault Tolerance in Mobile Agents

Rajwinder *et al.* [92] presented a mobile agent fault tolerance systems technique that allows a group of special agents to monitor events occurring in different hosts to avoid failure. This technique uses a prediction approach to detect precarious agent host so as to avoid failure. Monitoring agents identify faults in the system and then assist in recovering faults when an unexpected situation occurs. It offers an algorithm used to calculate host dependent to update weights and determines the criticality of the host. This is done to consistently monitor and evaluate the performance of the host agent in order to improve system response. The following mechanism is used to evaluate and monitor the entire system. (i) Monitoring Agent (MoA) – this type of agent monitors the global view and collects messages log information during execution, it serves as low level component. (ii) Manager Agent (MaM) – Manager is responsible for controlling the agents during monitoring, it also saves information of the checkpoint and controls the entire agent system. It serves as a high level component. (iii) Alerting Agent A*l*A – Alert Agent is responsible for alerts when fault and alarm information are detected, this is managed by Manager Agent. Checkpointing is used not only to check vulnerability but also to increase FT. From the experiment results discovered, [92] concluded that the proposed technique for monitoring distributed applications can improve FT, so the aim of this approach is to improve FT and ensure that MA reliability is increased. Checkpointing technique is used to offer fault tolerance. When the technique is platform independent the connection between MA is efficient. It uses message exchange for the agent to interact and messages exchanged from host to host defined as weights. The host can be defined, detected or prevented by the number. This proposed mechanism decreases vulnerabilities and increases FT in the system.

J. Agent Based Fault Tolerance in Wireless Sensor Networks

Sutagundar *et al.* [93] presented an MAFT mechanism that handles Fault Tolerance. This technique is a multi-agent Wireless Sensor Network FT based system that supports the implementation of multiple mobile agents. This technique offers a scheme that supports the estimation of faulty behavior on WSN in order to implement FT on (i) *cluster hierarchy level*, (ii) *node hierarchy level*, and (iii) *sink hierarchy level*. FT is built on three levels. The major benefits of the proposed techniques are as follow: (i) Fault tolerance on cluster level: information of all nodes is collected on cluster level and the data received gets analyzed together with all other data from other nodes and the difference is realized. The advantage of this strategy is that, FT on cluster level can be easily attained by removing incorrect data from nodes. (ii) Fault tolerance on node level: the node gets only samples which are correct, incorrect samples are removed and the node then works out the average of the entire correct samples. Data is sent to the sink node by means of cluster head. (iii) Algorithm for sink level fault tolerance: sink broadcasts active packets to all nodes and also duplicates the sink node. Then, the nodes send data to the sink node which was duplicated. This scheme is tested on the basis of node level faults, energy consumption, error, and cluster level faults. The aim of the proposed work is to enhance QoS on WSN, minimize malfunctioning, and minimize faulty data transmission and to detect failure on sensor nodes. Fault tolerance in [93] can be realized in two stages namely: (i) Fault detection: detection identifies nodes that are faulty in the topology of distributed WSN. (ii) Fault recovery: aims to recover fault by removing fault reading in distributed nodes setting.

K. A Fault-Tolerance Mechanism for Mobile Agent Systems

Leung *et al.* [94] presented a novel approach that prevents agents from faults. This approach copes with failure that is related to infrustucure and not failure that involves malicious actions. This approach is a centralized approach and checkpointing technique was choosen as a fault tolerance strutegy because it offers low bandwidth usage and low power usage. The technique is primarily categorised into three components namely; *Monitor Agent* (MA), *Worker Agent* (WA) and *Tracker Agent* (TA). Monitor Agent (MA) is denoted as an agent that monitors and manages status of other agents including TA and WA. Worker Agent (WA) is denoted as an agent that carries out the actual tasks (actual agent). Tracker (TA) is an agent that occupies on the third party host, it is liable for storing the migration path and authenticating the tasks carried out by the worker. As the WA migrates from host to host it stores its final state that can be retrieved and restored in case faults occurs as it moves.

When the crashed WA is recreated, the latest committed stage and information will be recovered and linked to the new created WA. The authors claim that as long as the WA can find a way to connect back to the WA, the system will not be blocked when the WA crashes in the future. Blocking can only occur if the host carrying the final state committed and the WA can be down simultaneously. The authors introduced two types of transition that were presented during simulation namely; locked and looped. This technique can guard the WA from failure irrespective of the increasing number of hosts it visits. For each stage looping was used to check the time spent, while locking was used to lock the WA when migrating; this will make it not do any task. If there is an error on the MA, this technique only allows the reconnection, which means that the MA cannot be recreated, this allows low power consumption. The proposed approach can reduce failures that can be introduced during network partitioning and minimize the chance of blocking to occur on the system. This approach offers efficient low bandwidth utilization algorithm when comparing it to other existing FT approaches which use replication mechanism.

L. Design and Evaluation of a Fault-Tolerant Mobile-Agent System

Lyu *et al.* [95] proposed a MAFT technique which adopted the approach presented on NAP [81]. The proposed approach takes into action two major agents that are responsible to monitor failure both on the system and on the agent, with a recovery mechanism. There are three significant agents that were adopted from NAP namely: *Actual agent*, *Witness agent* and a *Probe agent*. The actual agent plays the role of performing certain a computation, while a witness agent checks and monitors the state of the actual agent and the probe agent is responsible for recovering both the witness and actual agent when failure has occurred. Between the actual agent and the witness agent stands a peer-2-peer message forwarding mechanism that performs many tasks such as failure recovery or detection with a log that monitors and stores all the actions of the agents with advanced communication protocol between servers and agents. Many researchers have investigated various techniques to render Fault-Detection-Recovery (FRD), to restore servers that are off line, but if the agent is in the failed server during failure, most of the strategies fails to recover the agent [95].

The FT technique is designed in such a way that there are three tables that store checkpoints, logs and messages. Failure-detection-recovery mechanism is centred on the message logging and the message logging protocol is used mostly by the Witness agent to detect if the actual agent is lost or active. When the actual agent finishes executing, it creates a new witness agent

on the new server that it has migrated to and continues with its tasks. The actual agent spawns or creates a new witness agent on the current server. The process carries on until the actual agent (α) arrives at the final required destination. This approach offers an advanced approach that can restore a lost agent even in a case where the server containing the agents is offline. When there is failure on the server, it is possible to determine the point where the failure occurred to the actual agent assuming that there is no hardware failure. Simulation from this study indicates that their approach offers improved fault detection and fault recovery to mobile agent systems.

M. Fault-Tolerant Execution of Mobile Agents

Silva *et al.* [96] proposed a mobile agent fault tolerance technique built from scratch on JAMES, a platform developed in java [97]. The proposed mechanism is implemented based on three different components to support efficiency on Fault tolerance system namely; *Mobile Agents*, *Agent Manager*, *JAMES Platform Agencies*. Mobile agent uses lookup-directory to track agents, detect failed agents and track network partition. This technique offers enhanced network management by introducing a platform that can support the agencies and agent technology. The JAMES agents should be deployed in every host of the network with one or more Agent Managers (AM) and the JAMES agency platform offers a significant mechanism that enables the migration of agents. The Agent Managers are defined as entry-points of the system which supports the deployment of agent based network applications and offers management of running agents. For mobility control the mobile agents use the JAMES Application Programming Interface (API).

The idea of having more than one AM is to improve system availability so that if one Agent Managers (AM) can be compromised other the AM can continue running. This technique is catered for *FT on mobile agent*, *FT agent manager* and *FT Agency platform*. The mechanism is responsible for providing fault detection, data persistency, fault recovery and reconfiguration. Fault detection can be realized by using a simple protocol for diagnosing that allows the agent manager to monitor all the other agents and use messages to communicate their status such that the other will detect when one of the agent managers has faults. This mechanism presents lower overhead as compared to replication but offers great dependability. In this model, nodes can encounter failure and be recoverable. If the machine has crashed while the node is active, the agency will rely on the node to recover. This approach implemented various mechanisms and protocols to handle fault tolerance such as; atomic migration, recovery

protocol, agent fault detection, reconfigurable itinerary, mechanism for handling blocking of agents, a scheme to handle network partitions.

N. An Approach to Rollback Recovery of Collaborating Mobile Agents

Osman *et al.* [82] presented a mobile agent fault tolerance technique which is based on check pointing and offers a solution to exactly once issues by making it compulsory for the agent to checkpoint if there is committed status or update from external service. This mechanism offers fault tolerance during the migration of an agent by using a protocol that has two levels of commit to ensure dependability of mobile agents. The configuration of this approach was based on agent FT practical requirements, i.e., how agents can recover and how it maps with entire networks enterprise that support agent based services. This technique adopted three significant components namely; *Place*, *Region* and *Recovery Manager (RM)*. *(i) Place-* is a place where the agent executes its code, located on an individual node that has an agent-software-platform and also resources that agents use when executing. *(ii) Region-* is denoted as a group of places that has the ability to host mobile agents that render certain agent service. *(iii) Recovery Manager (RM)* is deployed on each of the regions that have agent based applications that runs on a failsafe node executing fault-detection approach and begging agent recovery when failure has been detected. The reliability of the recovery manger can be accomplished using election voting approach [98] and replication approach.

In order to maintain the rollback recovery mechanism there is a need to implement a central recovery manager. Fault tolerance approach presented by [99] and [96] affects fault-tolerance transparency and this must be prevented at all cost. The solution presented in [82], proposes that all checkpoints must be executed on the FT layer at the place by putting into place exceptions handling during agent execution unlike in [99] and [96] where the programmer hard coded the checkpoint methods on agents' code. This proposed mechanism offers FT during Inter-Agent interaction and communication and aims to solve three major faults that affect agents during execution, transient faults, place crash due to node failure and failure during migration due to before the agent commits.

3.6 Discussion

Computing entities such as application services, computer hosts and software agents are becoming mobile and decentralized. As the growth and development of MA technology acceptance increases exponentially, it is that imperative mobile FT approaches that can make

the system handle critical faults are incorporated to ensure that they are less prone to errors. FT is a major challenge that requires tools and techniques to be solved and improve agent based application in order to advance the adoption of the MAs. This has led several researchers to propose the solutions for problem such as reliability of agents during execution. Based on the analysis we conducted, we found that most of the FT techniques are agent based and can still be prone to failure that cannot be recovered. Furthermore, each technique analyzed has its own advantages and disadvantages. However, several of the techniques have no mechanism to handle parameters such as synchronization and security, and do not follow the exactly-once property. Consequently, there is still a need to introduce FT techniques that can handle these challenges which are not addressed in the current techniques.

In this analysis, we have learnt the importance and techniques of agents' communication which is essential or critical to fault detection and recovery on the network. These FT techniques and mechanisms can be adapted to design an effective FT system for SDWSN. In particular, the FT Manager (FTM) in [89] can be adapted to handle FT on the controller and the sensor node. FTM is also responsible for detecting failure on the sensors, as well as a FTM *Backup* which primarily monitors the FTM and can take over once it detects failure of the original FTM. On the other hand, the FTM will be responsible for using heartbeat techniques to monitor the controller for faults or failure. The detection of path failure is critical, whether the path consists of a single link or multiple links. Unlike replication schemes, a static 3-phase controller will be adopted to ensure FT in the case of faults of the executing controller. Other techniques that can be utilized are the voting technique, checkpointing and so on. In a nutshell, our proposed FT framework will be based on the idea of the Chameleon approach [89].

3.7 Chapter Summary

In this chapter different existing FT strategies and mechanisms that handle FT in MA agents were analyzed and reported. Based on the analysis, an approach was identified on which our proposed FT framework in SDWSN controller can be based to handle FT. This framework is discussed in the Chapter four.

Table 3. 1: Analysis of FT techniques in MA

Parameter	FT Approaches													
	[80]	[81]	[84]	[86]	[87]	[89]	[90]	[91]	[92]	[93]	[94]	[95]	[96]	[82]
Faults Recovery	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Fault type	N&A	A	A	A	A	A	A & S	A	A	A	A	A	A&N	A
Fault detection	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Exception handling	YES	NO	NO	YES	NO	YES	NO	YES	NO?	NO	NO	NO	YES	NO
Blocking/Unblocking	YES	NO	YES	YES	NO	YES	NO	NO	YES	NO	YES	NO	YES	YES
Central Management	NO	NO	NO	NO	NO	YES	YES	YES	NO	NO	YES	NO	NO	YES
Synchronization	NO	NO	NO	YES	NO	YES	NO	NO	NO	NO	NO	NO	YES	NO
Security	NO	NO	NO	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO
Performance Analysis	NO	YES	YES	YES	YES	NO	YES	NO	YES	NO	NO	NO	YES	YES
Energy efficiency	NO	NO	NO	YES	NO	NO	NO	NO	NO	YES	YES	NO	NO	NO
Replication	YES	YES	YES	NO	NO	NO	YES	NO	NO	NO	YES	YES	YES	YES
Checkpointing	NO	YES	NO	NO	YES	NO	YES	YES	NO	NO	YES	YES	YES	YES

***A=agent, N= node & S=server.

Table 3.2: Advantage of each FT Technique

Ref.	Advantages
[80]	This technique offers low CPU capacity during FT execution and it minimises blocking at all costs, the agents are never in the same node making recovery to be possible always.
[81]	No additional processors are required and no programmer defined recovery is requires during MA failure recovery.
[84]	This technique uses a voting protocol which is based on the FT majority quorum algorithm.
[86]	MoCA offers User transparency and agent failure can be handled using both the forward and backward recovery scheme
[87]	In this approach, checkpointed scheme parallel offered to enforce fault tolerance using antecedence-graphs.
[89]	This approach is adaptive because during run-time agents can be removed or added and it uses the backward-recovery approach. It also support synchronization parameter, whereas all other techniques do not cater for this parameter.
[90]	This approach offers significant advanced performance and reliability. The time checkpointing, total trip time and migration time are used to evaluate improved performance.
[91]	This approach is adaptive is scalable as the message logs cannot be too large and it consumes low bandwidth.
[92]	This proposed mechanism decreases vulnerabilities and increases FT in the system making the systems more reliable.
[93]	This scheme offers advanced energy consumption and can cater FT on a cluster level.
[94]	It offers efficient low bandwidth utilization algorithm when comparing it to other existing FT approaches which uses replication mechanism. Too much bandwidth is utilized when using replication schemes due to sending the agent's state to replicas and complex commit protocol between replicas.
[95]	The advantage of this technique is that; in case of failure the system has a roll-back mechanism to abort actions that are not committed [82] and check-pointed data to restore witness and actual agent that are lost.The actual agent uses checkpointing for computation.
[96]	This technique allows software rejuvenation [100], allowing agency to refresh its state when the system has low resources (such as memory or threads) then there is a possibility to refresh the state of that Agency.
[82]	This approach offers exception handling, replication scheme, election voting mechanisms and centralized RM.

NWU LIBRARY

Table 3. 3: Disadvantage of each FT technique

Ref.	Disadvantages
[80]	The replication scheme introduces overhead that increases the size of the agent and the number of stages.
[81]	The loop hole of this approach is that, it cannot support or handle partitions in the network. If a MA terminates, the NAP will also terminate.
[84]	The disadvantage of this technique is that, if it happens that data manager gets suspended, the agent also gets suspended. This technique has a high bandwidth consumption when sending the result and during the process of commit.
[86]	Although MoCA supports various techniques for handling fault tolerance in the network, their techniques might not always be applicable to other challenges.
[87]	The disadvantage of this technique is that, it expands the overhead on every log message and antecedence graph message [101].
[89]	The drawback of this approach is that, it doesn't differentiate failures between the network and the node, i.e., switch failure, link failure. Another drawback of Chameleon is that there is extreme blocking if errors occur during nodes execution
[90]	The disadvantage of this technique is that it can sometimes interrupt the exactly-once-property of MAs.
[91]	This technique is not adaptive to other circumstances and platforms.
[92]	The disadvantage of this technique is that apart from the parameters that have been considered, some other parameters can be considered in order to accurately measure host vulnerability and to gauge efficiency such that more formal model can be built.
[93]	Fault tolerance on cluster level can be easily attained by removing incorrect data from nodes.
[94]	One of the disadvantages of this approach is that rate of location or place is high, the chances of the worker agent to survive is low.
[95]	Failure on this technique can occur if the actual agent gets lost when leaving or heading to the server or when the actual agent gets lost with or without logging.
[96]	Failure on the agent can occur when the agent migrates between hosts or nodes; failure can occur when the agent can find on network partition, when the agency can crash and when there is software bug.
[82]	The shortcoming of using centralised recovery mechanism is communication bottleneck and can incur overhead during agent execution or recovery. There are bottlenecks when using centralize recovery manager

CHAPTER 4

SDWSN Fault Tolerance Framework

4.1 Chapter Outline

This chapter discusses FT strategies for an OpenFlow-based SDWSN controller. This is important to enable the network to remain operational in the event of a fault on the network. However, the objective is to use the knowledge gained from chapter 3 (the analysis of MAFT approaches) to design of a fault tolerant SDWSN controller. This chapter begins with a brief introduction, a justification the choice of FT strategy and progresses to the proposal of a SDWSN FT framework (FTF). Moreover, different components of the proposed framework and its operations will be addressed.

4.2 Overview

SDWSN is an evolving networking paradigm that uses dedicated software to allow decoupling of forwarding planes and the control plane. SDWSN can be disposed to failure that can interrupt the entire network operations. In particular, the OpenFlow-based SDN controller which is the brain of the network poses as a single point of failure and in the event of controller crashes, the entire network will fail. Consequently, there is a need to implement an effective FT strategy to ensure the controller is fault tolerant in the face of adverse network conditions. That is, to enable the controller to detect, manage and recover from faults on the network. Although SDWSN controller is inbuilt to handle fault tolerance, it can also encounter failure that cannot be resolved [40]. In this case, the SDWSN fails to automatically recover from the network faults that affect it. In addition, Lu et al. [102] reported that failures in SDWSN are also caused by node failure. Typically, the wireless sensor nodes have limited energy to communicate and for processing and are always left with very limited amount of energy. However, since SDWSN is specifically designed to support network programmability, these types of challenges can be achieved by designing FT mechanisms that can be integrated into the controller to detect and recover from faults in the event of failures. As such, a FT strategy must be designed to focus also on being lightweight in energy utilization, resources and communication ensuring good Quality of Service (QoS).

Thus, the main goal of this chapter is to propose a FTF for SDWSN to ensure the reliability of the controller execution which in turn will ensure that the wireless sensors remain functional

regardless of the faults that affect the controller. Throughout this chapter the word "controller" will be interchangeably used with the SDWSN controller, and FT strategy will represent SDWSN controller FT technique. FT can be delivered through the use of dedicated software, hardware, or a dedicated combination of both. The disadvantage of offering FT through dedicated hardware is that it will provide a static FT that cannot be changed throughout, such architectures can only be oriented to specific applications. Most distributed network environments support the use of software-based solutions to offer FT. To this end, our FT approach will be software-based. This is necessary because of the programmability of the controller, making it easy to integrate FT mechanism in the control plane to enable effective fault detection and recovery in the network.

4.2 Choice of FT Approach

Our proposed FTF will adopt the approach of the *Chameleon* FT approach [89] discussed in chapter 3. In chapter 3, we evaluated existing MAFT techniques to identify FT approaches that provide efficient FT approach in mobile agents. Based on the findings and the characteristics of each MAFT technique studied, this research concluded that the FT strategy of *Chameleon* [89] has a suitable FT mechanism that can be employed to design and develop an efficient fault tolerant OpenFlow-based SDWSN controller. The adopted strategy [89] exhibits a powerful FT technique by way of detecting and recovery operations and can be easily reused to design an effective and robust fault tolerant in distributed networks such as the SDWSN. Moreover, it supports both permanent faults and temporary faults and offers checkpointing and recovery. Thus, *Chameleon* [89] FT techniques support the availability of various requirements levels to be offered simultaneously on the networking environment using special agents. In addition, it operates with specialized agents that are capable of detecting failure and recovering from failure quickly. The specialised agents are the FT *manager*, *daemon*, and *common* agents [89]. It supports the concept of manual and automated agent engineering and re-engineering [89], which allows the creation of a new agent required to handle a specific functionality. Agents on the *Chameleon* FT approach were specifically designed to ensure that there is no single point of failure on individual agents [89]. Based on its acceptable performance operations in ensuring strict consistency and fault tolerant system for agents' execution, we therefore choose this FT approach to be used to propose an effective OpenFlow-based SDWSN FTF to eliminate the threats posed by the single point of failure associated with the OpenFlow-based SDWSN controller.

4.3 The SDWSN Fault Tolerance Framework

This section presents conceptual designs of the proposed FTF in SDWSN. The focus is on the architecture of the FTF and its components. In addition, the different types of faults to be handled will be explained as well as the algorithms for fault detection and recovery. The FTF proposed in this research is aimed at enabling the network to remain operational despite the existence of faults in the controller. The essence is that, as the controller is the brain of the entire network, it is imperative that it is equipped with FT mechanism to ensure the single point of failure it possesses is eliminated. In this case, the controller will remain fault transparent in the detection and recovery from failure in the event of faults of any kind. That is, in the event of fault, the system or system components will continue to be operational as well as recover speedily from the faults.

4.3.1 Types of Faults to Detect

There are several types of faults that can cause the controller to fail or cause the entire SDWSN to fail. In this case, our FTF will be designed to handle such faults in the events they manifest and ensure that the controller remains operational. The various faults are discussed as follows:

A. Abnormal termination fault

Abnormal controller terminations are defined as undesirable terminations that the controller undergoes during its execution. If such activities and behavior occurs, it jeopardizes the flow of the operation on the network. During abnormal terminations, the controller doesn't completely crash, it only terminates during operation and interrupts the flow of packets. Abnormal termination will affect the controller temporary stop operating, packets lost, interrupt fault detection, the routing protocols and so on. To guard against these forms of faults, the proposed FTF provides a FT module to handle abnormal termination faults.

B. Controller crash fault

The crash of the controller poses a scenario or a faulty situation where the controller crashes and stops working. That is, controller crash emerges when the controller stops working and exits. Typically, the controller can also crash due to software bugs that were not detected before deployment. On a single controller like the OpenFlow-based SDWSN where the controller is a single point of failure, in the event of controller crash, the entire network will fail.

C. Livelock/Deadlock

Deadlock is defined as a state in which SDWSN controller is unable to take action because the controller is waiting for a request that was sent to be executed or the controller has executed multiple process that competes for the same resources and both are continuously denied to accesses the resource in order to complete the required task. On the other hand, Livelock is denoted as a scenario whereby a process or a job which is requested to operate is not progressing, that is, there is no progress made on the controller.

To guard against this fault, the FTF is designed with a Livelock or Deadlock detection mechanism that can be triggered when the controller has encountered such faults. The mechanism includes deadlock prevention, deadlock detection and deadlock recovery [98].

D. Incorrect Result

The FTF is also designed to cater for a mechanism that will help the SDWSN controller against voting for incorrect results that were received after execution. Proposing such a mechanism will enable the SDWSN controller to handle erroneous computations during execution.

E. FTM failure

Although the FTM is the module that handles FT execution and activities in the FTF, it is possible failure can take place during the execution of the FT techniques especially by attacks of any type such as DoS. In this case, it is important to have a mechanism that can detect faults that can trigger such failure during its execution. The FTF also has a mechanism that handles faults of this nature that guard against the failure of the FTM.

F. Stable storage failure

The stable storage failure is a failure that can occur when the stable storage maintained by the checkpoint manager fails. The stable storage stores all check-pointing information which is the state of the executing controller which can be used for recovery or roll-over when the controller fails as well as other important decisions. However, if the database fails, the controller won't have any source of information to recover faults. Therefore, a mechanism is designed to detect this fault and act accordingly to thwart it.

4.3.2 Fault Tolerant Execution Cycle

In the FTF, to ensure effective fault detection and recovery we proposed three execution stages that are important to ensure the controller is fault tolerant. The processes necessary for FT

execution on the FTF are as follow: fault detection, recovery planning phase and lastly the recovery. (See Fig. 4.1). The execution stages are discussed as follows:

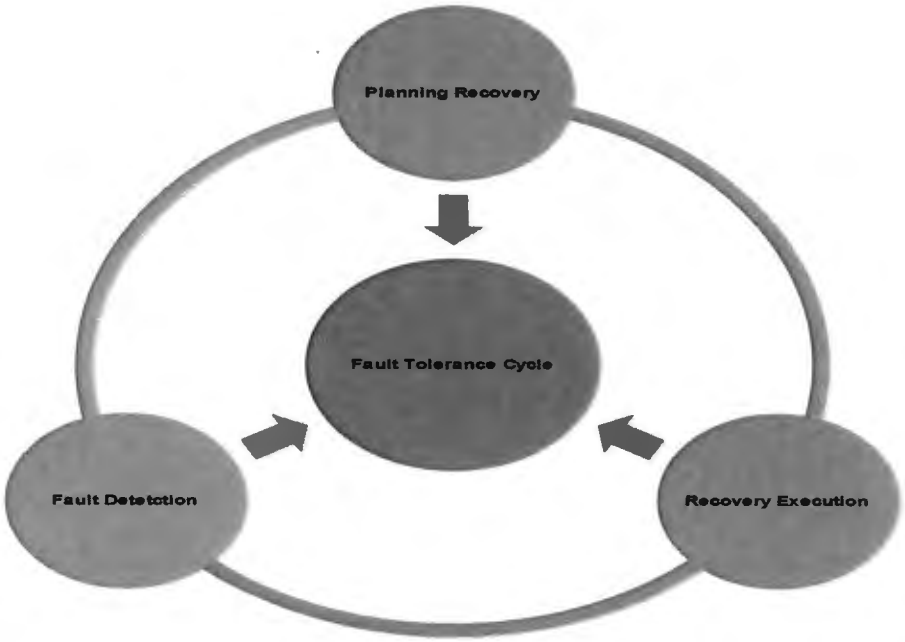


Figure 4.1: Cycle for Fault Tolerant Execution [103]

A. ***Fault detection:*** This stage in the FTF is used to detect the manifestation of specific faults that occurred on the SDWNS controllers. In the FTM, there is a module, possibly the Heartbeats Manger that is designed to detect all forms of faults at this stage to enable it to plan for recovery as soon as possible.
B. ***Recovery planning:*** Once the faults are detected either on the controller or the FTM, the next step is to plan for effective recovery. Planning here involves diagnosis of the type of fault involved and the initiation of the necessary plan to recover from such fault.
C. ***Recovery execution:*** Once the recovery planning has been executed successfully, the final stage is the recovery execution which is one of the complex processes during FT execution. In this case, based on FT strategy selected to recover the detected fault, recovery execution will initiate or execute complex configurations that will fix detected faults and enable normal service continuation.

These stages will be discussed in details as they apply to the FTF proposed in this research. In the FTF, "FT strategy" will be represented by FTM. FTM is the manager of FT execution in the framework that ensures the controller remains fault tolerant to keep the network operational. It offers a mechanism for improving FT on the network and

architecture designed in a manner that allows the network system to continue with normal operation in the face of failure of some components and subcomponents. This is accomplished by incorporating exceptional conditions into the FTM. In general, FTM is designed to ensure that the controller is not a single point of failure in the SDWSN.

4.4 Proposed FT Strategy Design

In this section, we present the design of our FT approach to make SDWSN controller fault tolerant. The approach is based on the idea of the Chameleon approach [79], but the design is what distinguishes them. In this case, we proposed a FTM that oversees the affairs of FT execution which is a new module we proposed in the control plane of the SDWSN. Our FTM is not integrated into the controller but sits outside the controller in the control plane unlike the Chameleon that embeds FT into the MA. The details of the design and analysis of each component involved are discussed in the subsections that follows:

4.4.1 3-Phase Controllers

The existing OpenFlow-based SDWSN controller is single and has the responsibility of providing an abstract or global view of the entire network, monitoring, managing and controlling the entire network. However, in the event of faults or attacks on the single controller, the whole network will fail. Implementing a centralized controller has challenges of having a single point failure which affects both reliability and availability, so it is important to introduce more than one controller to cater for fault tolerance. Thus, to offer FT on the SDWSN controller, we proposed a 3-phase controller called the EDD controller which are the *Exousia controller*, *Deamon controller* and *Dunamis controller*. See Figure 4.2.

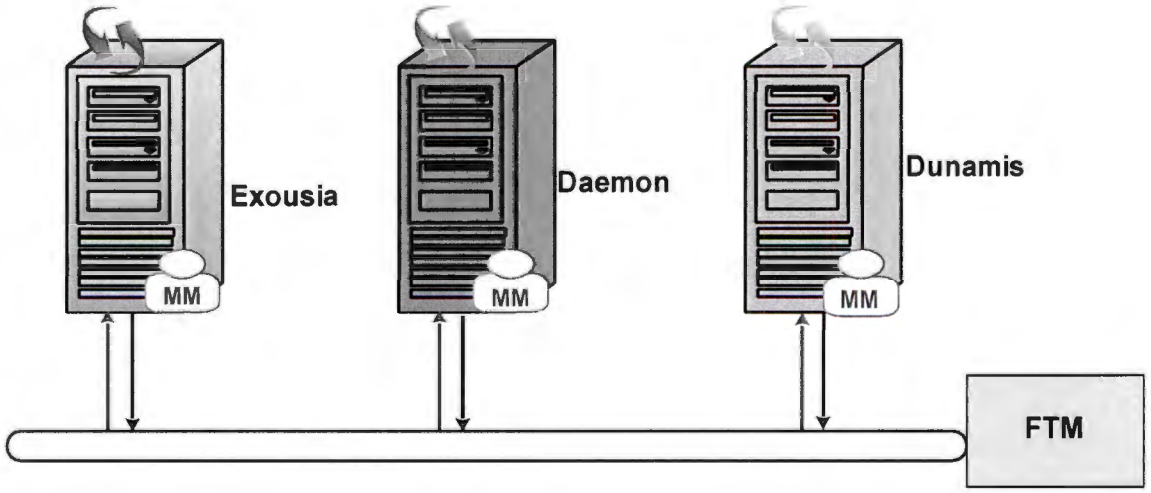


Figure 4.2: EDD controllers

With these three controllers, only one controller will be active and functioing at a time to ensure exacly once property of FT execution [84], though the other two controllers have the same functionality that the active controller has. In this case, the Exousia controller will be the main controller functioning in the network in the absence of faults or attacks while the *Deamon* and the *Dunamis* controllers serve as intelligent standby or back-up controllers ready to take control of the operation in case the Exousia controller is non-functional. The EDD controllers are defined as the primary controllers that are responsible for managing flow control and offer intelligent networking on a SDWSN. These controllers function interchangeably with the aim of ensuring controller fault tolerance, reliability and availability.

All three controllers communicate with the FTM by receiving and transferring messages from the heartbeat manager using the Message Manager (Msg_Mnr or MM). This is important to effectively monitor the controllers and the network for potential faults or attacks. The FTM is programmed to offer strict rules and procedures on how to hand over operations to the back-up controllers (*Deamon* and the *Dunamis* controller) when the main controller is found not functional. Moreover, during the event of failure, the FTM uses the voting technique to determine which controller should replace the faulty one while the main or Exousia controller recovers.

A. Message Manager (Msg_Mnr).

As shown in Figure 4.2, each has Msg_Mnr or MM which is responsible for communication interface between the controllers and the FTM. In the controller, MM also monitors the health and reports its status through Heartbeat message exchange to the FTM.

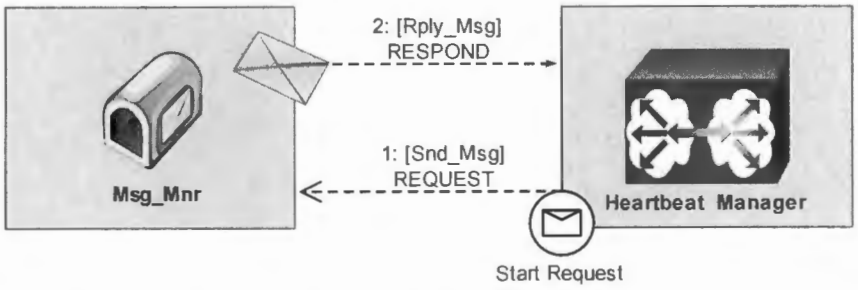


Figure 4.3: Msg_Mnr and Heartbeat communication

At every interval, say 60 secs, the Heartbeat Manager will request information from the controller to check its state. To this end the Msg_Mnr will reply to the Heartbeat Manager. The reply content of the MM is the state and the health of the controller. However, if Msg_Mnr fails to reply to the Heartbeat Manager with the stipulated timeout, the heartbeat stores the error into its temporal database and then informs the Execution Manger to initiate the fault tolerant strategy using an alternative controller. Figure 4.3 shows the message exchange between the

Heartbeat manager and the Msg_Mnr to check if the controller is active or dead. As shown in Figure 4.3, when the Msg_Mnr sends controller information to the heartbeat manager, they send it together with their current state and health status. This content is only stored in temporal storage of the heartbeat manager only to be used during the voting process to decide which controller is suitable to replace the failed controller.

4.4.2 Fault Tolerance Manager Framework

Unlike the Chameleon FT strategy [89], this research proposes a FT strategy that is placed externally in the control plane having the capability to configure and manage distributed controllers to operate efficiently on the SDWSN, environment and adheres to the properties of exactly once and non-blocking of FT execution. Our FTM consists of a group of managing modules called Managers that invoke different functions to FT on SDWSN controllers (EDD controllers). FTM has the responsibility of monitoring the controllers to effectively detect failure and recover from it on the network. Figure 4.4 capture the managers that are available in FTM to provide FT.

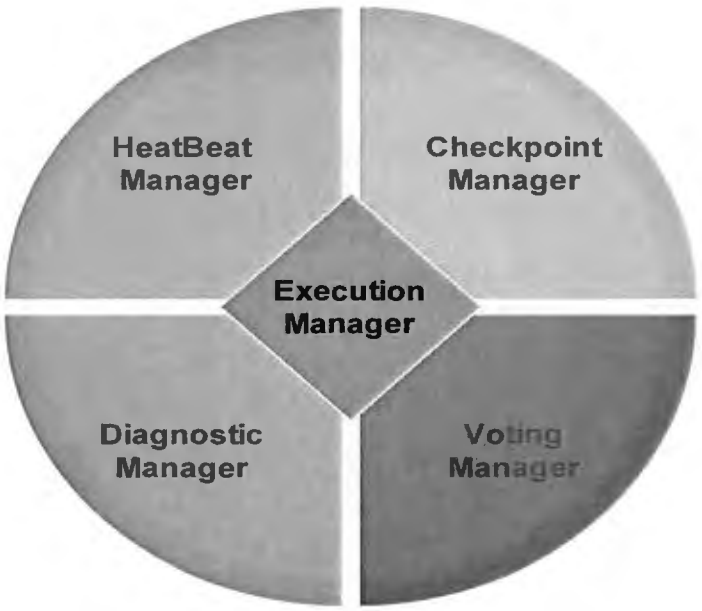


Figure 4.4: The FT managers

The main objective of the FTM is to enable the network to remain fault transparent, detect when failure occurs on the network by using a powerful mechanism to keep the network operational as well as to recover from the failure. This FTM manager is expected to be reliable and highly available to detect and recover faults quickly. That is, it is designed to ensure strict consistency and fault tolerant systems. The design supports more than one controller but only

one or a single controller will operate in the network at a time, thereby giving the controller a global and constant status of the network. The FTM is shown in Figure 4.5 alongside its components which are the managers.

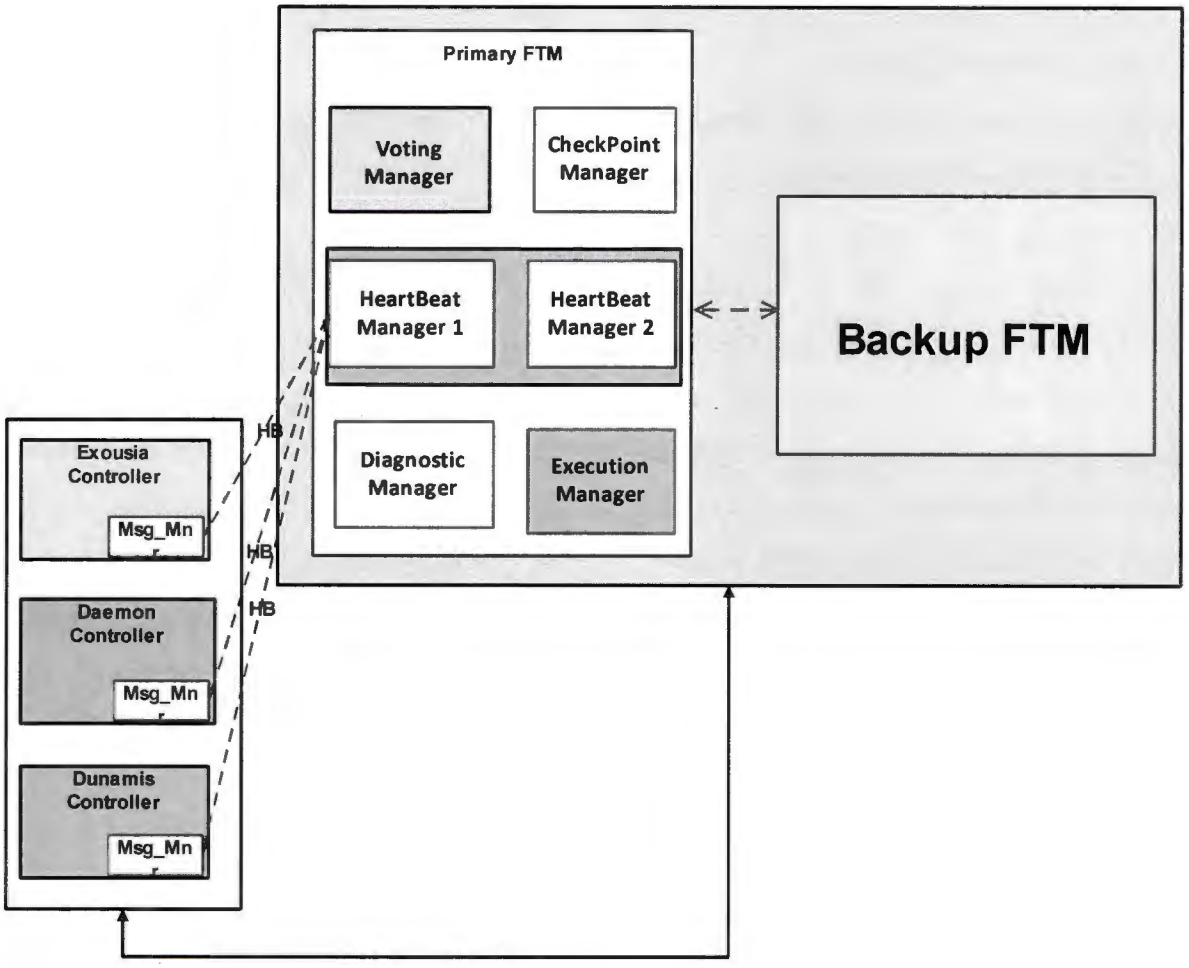


Figure 4.5: The FTM

In a nutshell, the FTM is responsible for monitoring over the execution of each controller to detect and recover from faults. If a fault is detected on the controller, the FTM techniques will invoke the appropriate FT strategy to recover from the fault or failure. The FTM technique is considered robust as it is designed to manage and detect faults on the system by remotely executing protocols that enable the network to remain operational without failing in the face of faults.

Furthermore, FTM is not immune from failure. Thus, we also proposed a backup (Backup FTM) as shown in Figure 4.5 which sits on the same control plane. To ensure consistency in the state of the FTM, it synchronizes with them at regular intervals and has all the components of the FTM. In Figure 4.5, the FTM offers special FT mechanisms or managers: *Heartbeat,*

Execution, Voting, and Checkpoint manager. These managers play significant roles in ensuring FT in the network and the EDD controllers.

Details about the managers are discussed as follows:

a) ***The Heartbeat Manager***

The Heartbeat (HB) Manager is responsible for monitoring and querying the status of the controllers to determine if they are active or dead. In Figure 4.5, there is *Heartbeat1* and *Heartbeat2.*

*Heartbeat*1 monitors the EDD controllers by communicating with each MM to collect information about them to detect if they are alive or dead as well as the health status which will assist the Voting Manager to make important decisions. In this case, the Heartbeat 1 maintains a table (local storage) called the Heartbeat Manager Table to store information about the EDD controllers, the information will include the controller ID (C_ID), response, timer, state and the health status of the controller. The C_ID is a unique ID of each controller, the response indicates whether the controller responded or not. Yes (Y) will be used for a response received from the Msg_Mnr, otherwise NO (N) will indicate no response after a given TIMEOUT which signifies fault detection. The state will be the current state of operation of the controllers which has a reference to the stable storage. Moreover, the timer stores the time each response was received while health shows which controller is fit and in good state (see Figure 4.6).

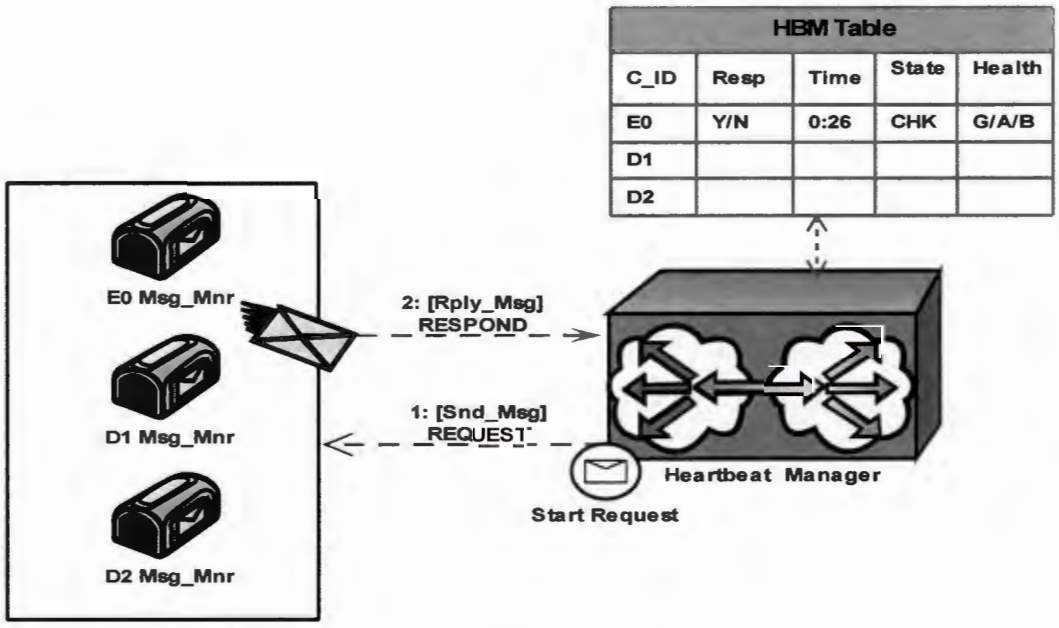


Figure 4.6: Heartbeat manager

Accordingly, Heatbeat2 is use to monitor between the FTM and the Backup FTM to communicate as well as synchronize with each other. In this case, if the Backup FTM did not receive a reply or heartbeat from the FTM, it automatically takes over from the FTM after a stipulated TIMEOUT. In summary, Heartbeat1 monitors failure of the controllers while Heartbeat2 monitors failure of FTM.

b) Diagnostic Manager

the strategies to invoke for such fault. That is, it is responsible for identifying and analyzing the magnitude of the fault detected and determining what should be done to recover the detected fault. It lays a proper recovery plan upfront before recovery execution. The output of the diagnosis phase is not only to plan for recovery, but correspondingly informs that controller about the type of faults that has occurred so that the controller can take accurate actions to solve the diagnosed fault. The diagnosis state is composed of sub-components such as plan for recover and inform FTM.

c) *Execution Manager*

Execution Manager is in charge of executing the FT strategy once a fault is signaled by the HBM. In the event of a fault and depending on the nature of the diagnosis, the Execution manager is the one responsible for initiating voting in the Voting Manager as well as the Checkpoint Manager. The Execution Manager specifically monitors the HBM with full supervision during execution in order to take correct action immediately. For example, if the detected fault requires the rollback technique to recover, the Execution Manager will automatically initiate the recovery module by sending a signal to the Checkpoint Manger to execute in order to recover from the fault.

d) Checkpoint Manager

The Checkpoint Manager as shown in Figure 4.7 is responsible for taking checkpoint which is the snapshot of the EDD controllers state at a given interval and recovery from fault or failure via rollback. It has two modules namely: The Checkpoint module and the Rollback module. To ensure communication overhead is not incurred, at every interval of 60 seconds, a snapshot of the executing controller's state is taken using Checkpoint() which is stored the state of the network in a stable and reliable storage.

Moreover, when a failure is signaled or detected by the HBM, the Checkpoint Manager will use the latest checkpoint by initiating the Roll_back() to execute recovery on another controller

(D1 or D2) after voting. Figure 4.7 demonstrate an algorithm followed to execute Checkpoint() and Roll_back().

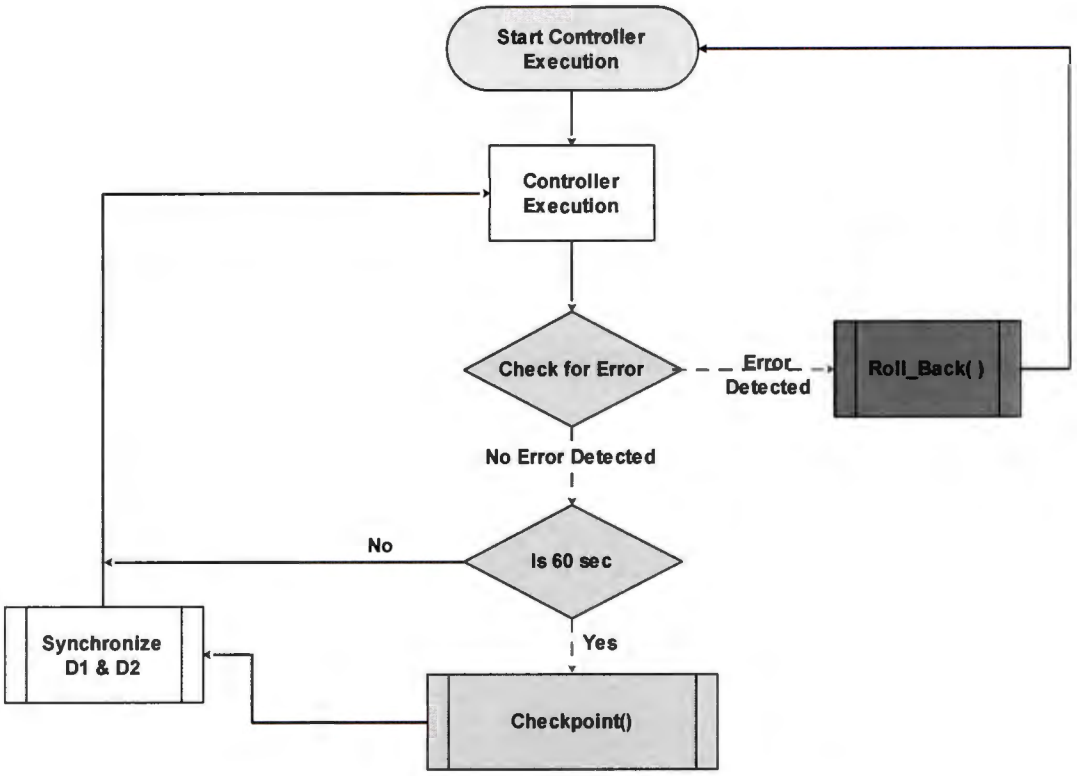


Figure 4.7: checkpoint and rollback algorithm

When Checkpoint() and Roll_Back() is complete they send Checkpoint() and Roll_Back() acknowledgment to the checkpoint manager. However, to enhance efficient utilization of resources, when a new checkpoint is stored in stable storage, the old checkpoint automatically gets deleted. In this case, the state of the Exousia controller will automatically be synchronized with other standby controllers. Figure 4.8 presents the process involved in the checkpoint and rollback and the stable storage.

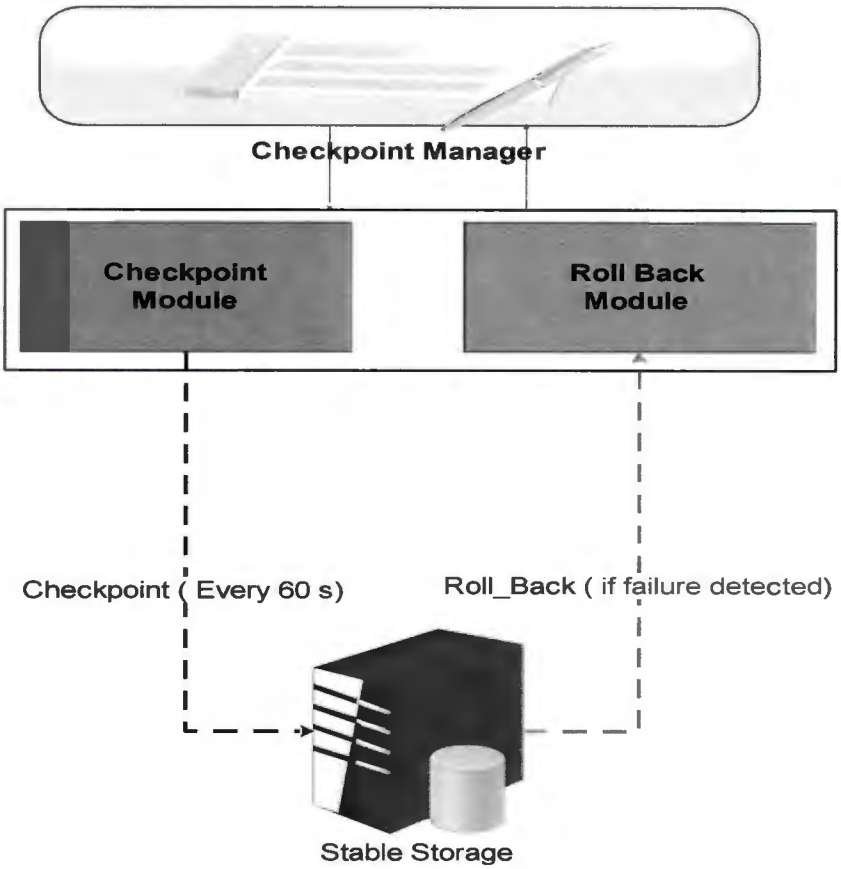


Figure 4.8: Checkpoint Manager stable storage

To ensure consistency of the network state when the new controller takes over, the stable storage will synchronize its state with the standby controllers while still maintaining its content. Thus, to ensure that there is efficient use of resources and avoid delayed communication or resource overheads, checkpointing will be carried out every 60 seconds as shown in Figure 4. 7 and 4.8. Clearly checkpointing will differ depending on the amount of space available to store the state of the network as well as its bandwidth.

e) Voting Manager

The Voting Manager is responsible for executing voting strategies when a fault or failure is detected using the results from the Heartbeat Manager and Checkpoint Manager. To this end the Voting Manager is used to select an appropriate backup controller if the executing controller fails. To select a standby controller to take over, the Voting Manager will check the controller's current state and the health status registered in the HBM Table shown in Figure 4.6. In this case, it will compare the state of the controller with the latest checkpoint taken. The controller with the latest checkpoint will be the one selected to take over in the event of failure.

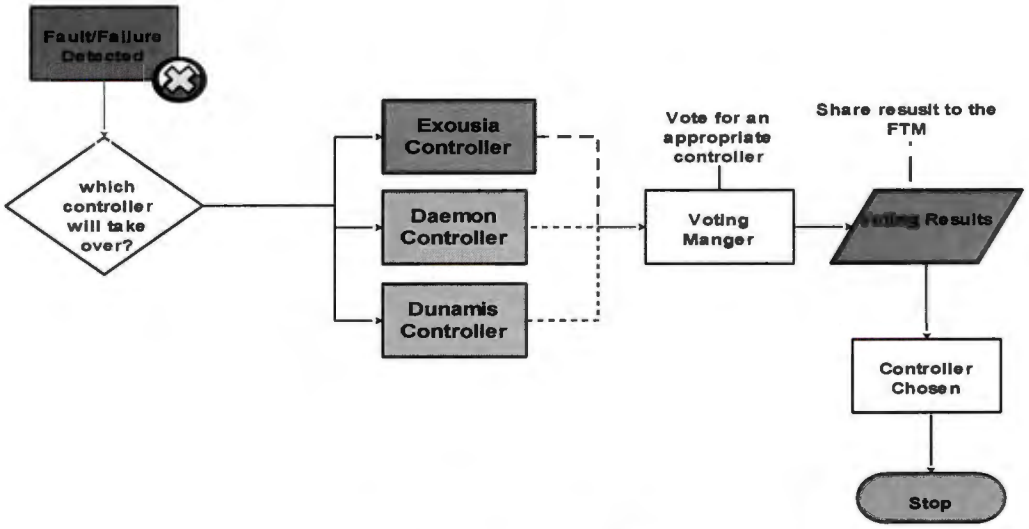


Figure 4.9: Voting Manager Execution

The algorithm for the voting is shown in Figure 4.9. As shown, the Voting Manager will compare different available data regarding the health status and the current state of the controller and based on the information gathered, the Voting manager will select the appropriate controller to continue network operation.

f) Backup FTM

The FTM consists of a backup of itself (Backup FTM), the Backup FTM is responsible for detecting a fault on the FTM with the intention to preventing the primary FTM to be a single point of failure by initiating recovery technique by enabling itself to take over from the FTM by Voter technique. The Heartbeat technique is used to detect failure of the FTM. The Backup FTM has the same components of FTM, and it follows the exact same process explained in Figure 4.5.

4.5 Fault Detection and Fault Recovery

One of the important requirements of our FT technique is fault detection and fault recovery. However, existing OpenFlow-based SDN have been designed to handle FT on the network [104] but not on the controller. When there is a single controller, if it is not made fault tolerant, the entire network will fail as it will not be able to resist failures. To manage this risk a FT framework is critical.

The FTF discussed in this research serves as a general guidance that can be used to implement a robust FT on the SDWSN controller. The proposed framework consists of two core

components: the 3-phase controllers and the FTM. This section discusses the FTF can detect and recover the controller from fault or failure.

4.5.1 Assumptions

In this section we discuss some assumptions that were made during the design of the EDD FTF.

Assumption 1: The controller can crash due to many faults as defined on Section 4.3.3.

Assumption 2: When the *Exousia* controller fails, the Backup controllers takes over, the selection of which Backup controller should take over is determined by the Voting Manager. Section 4.3.3 discussed the type of faults that the network is prone to.

Assumption 3: To ensure exactly-once property, only one controller executes at a time on network operations in order to avoid the same process to be executed many times by different controllers (see Figure 4.9)

Assumption 4: To ensure non-blocking, failure on the controller cannot block the other controllers from taking over the network and continuing with execution.

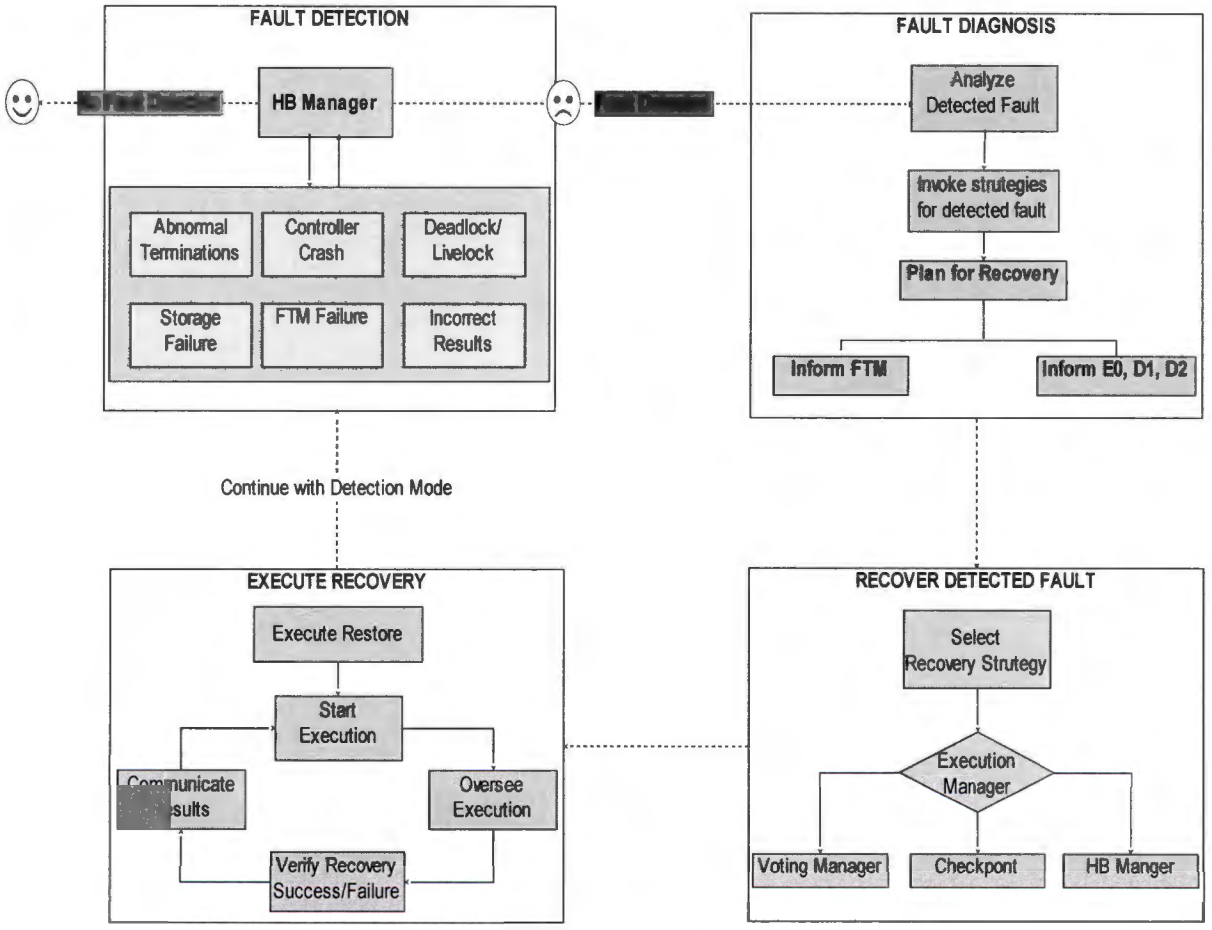


Figure 4.10: Fault detection & recovery cycle

4.5.2 Fault Detection and Recovery mechanism

This section describes how fault detection and fault recovery is achieved on the EDD controller using the proposed FT strategies and all the executions are managed by the FTM. An overview of how fault detection and fault recovery is achieved is represented in detail in Figure 4.10. Explanation of the framework execution stages that are followed during the operation of fault tolerance are as follows: Fault Detection, Fault Diagnosis, Recover Detected Fault, and Execute Recover.

A. Fault Detection Stage

Fault detection stage is the detection of the manifestation of faults on the SDWSN controller that can result in intolerable or undesired behaviors. Like any fault tolerant system, our FTF contains a fault detection component that can detect when specific faults have been detected. In this case, the HBM is the detection component responsible for detecting fault and report to the Execution Manager. If any fault is detected by the HBM, all information gets forwarded to the fault diagnosis component. Details of how the HBM operates is discussed and presented in Section 4.4.2. Figure 4.11 presents the algorithm for fault detection during FT execution.

Fault Detection Stage

```
1. Every 60 seconds check controller heartbeat () {
2. If (heartbeat == True){
3. Send information to the HBM Table
4. Continue operating }
5. Else if (!heartbeat== True)) {
6. Ping again ()
7. Wait 20 second for response ()
8. If (heartbeat== True){
9. Go to step 3 }
10. Else if (!(heartbeat== True))
11. {
12. Notify the FTM ()
13. Deactivate the Controller()
14. Prepare for recovery (){
15. Invoke voting_manager()
16. Invoke checkpoint() && rollback()
17. Activate backup_controller ()
18. } }
```

Figure 4.11: Fault Detection Pseudocode

B. Diagnosis Stage

At this stage a fault has been detected by the HBM, and here the fault diagnosis component is used to diagnose the fault detected to identify which fault it is to enable it invoke the strategies required to solve the issues prior to executing recovery. This component is also used to plan for recover, it analyzes the magnitude of the fault detected and determining what should be done to recover the detected fault. Correspondingly the Diagnosis Manager informs the Execution Manager and the controllers regarding the type of fault that has been detected, enabling the controller to take correct actions to solve the diagnosed fault. Plan for recovery process enables quick recover during the stage of recovery. It also dictates whether Rollback and Checkpoint will be required. The pseudocode in Figure 4.12 shows the algorithm involved in Diagnosis stage during controller failure:

Diagnosis Stage

```
1. get fault notification (from HBM) {
2. Analyze detected fault ()
3. Invoke required FT strategy
4. Execute a plan for recovery{
5. Notify the FTM ()
6. Notify the Controller
7. }}
```

Figure 4.12 Fault Diagnosis Pseudocode

C. Recover Fault Detected

Based on the selected recovery action, the Recovery component is responsible for executing the recovery process as planned from the Fault Diagnosis stage. For complex recovery, these components might require the most recent checkpoint that was taken during the process of Diagnosis stage. The recovering process is not done manually, rather it is monitored by the Execution Manager that enables the FTM automatically to execute the required FT action subject, to the fault detected and technique that was planned for recovery to be executed. During fault recovery the HBM Table is used by the voter to make accurate decisions. This is captured in Figure 4.13.

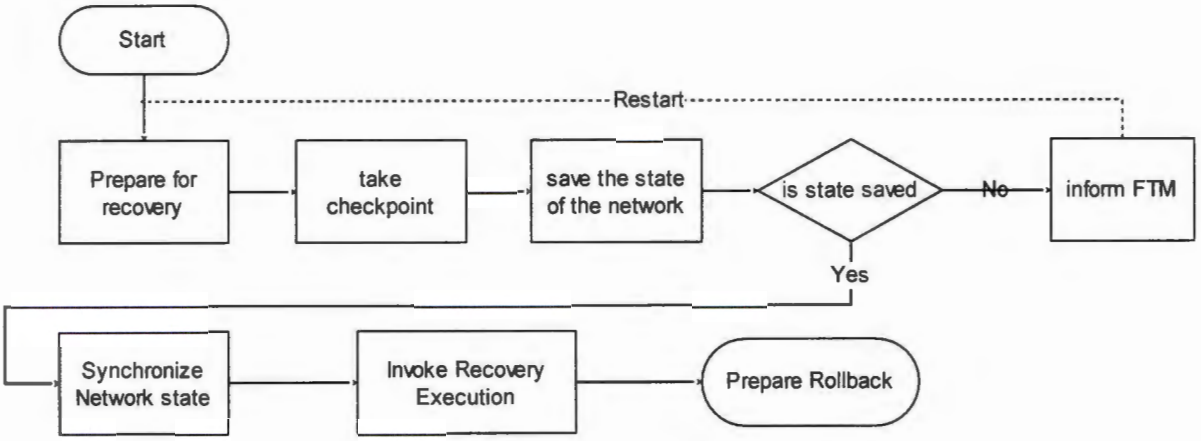


Figure 4.13: Recover fault Detection

D. Execute Recovery

The Execute Recover is the continuation of the previous stage. At this stage, the focus is on final execution of recovering failure. The Execute Recovery is the last stage of execution; it is composed of Rollback components which are mainly responsible for starting the recovery process. This is the stage where the corrective action mechanism is applied, this is a process to repair or recover the detected fault. It uses the Rollback mechanism to recover the state of the network in the controller. The Execution manager is responsible for overseeing the processes of recovery. When recovery starts, the Execution manager checks if recovery was successful or not and informs the FTM about the results. Note, after recovery the FTM still invokes the heartbeat manager to continue detecting on the network.

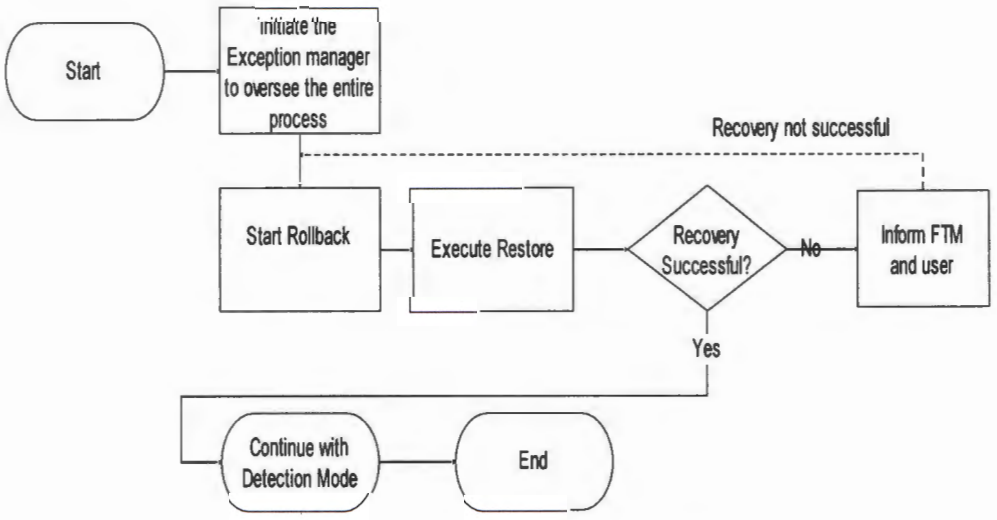


Figure 4.14: Execute Recovery

4.6 Controllers Synchronization and Network State Consistency

A controller is the brain of the SDWSN network [19], therefore it is advantageous to have multiple controllers that plays a role of providing FT on the network. The benefit of multiple controllers is to avoid the challenge of having the controller as a single point of failure. However, maintaining synchronization and consistency between these multiple controllers is a critical function. The FT Framework is designed to handle this requirement. For instance, when the Exousia controller is in charge of the network operations the Daemons *controller* or *Dunamis controller* only serves as backup ready to take control of the network in case the Exousia controller fails. This make it mandatory for the backup controllers to always have the latest state of the network. The checkpoint stable storage is configured to synchronize the current state of the network with the backup controller using the Checkpoint Manager. Figure 4.15 shows how the controller can synchronize its checkpoint contents.

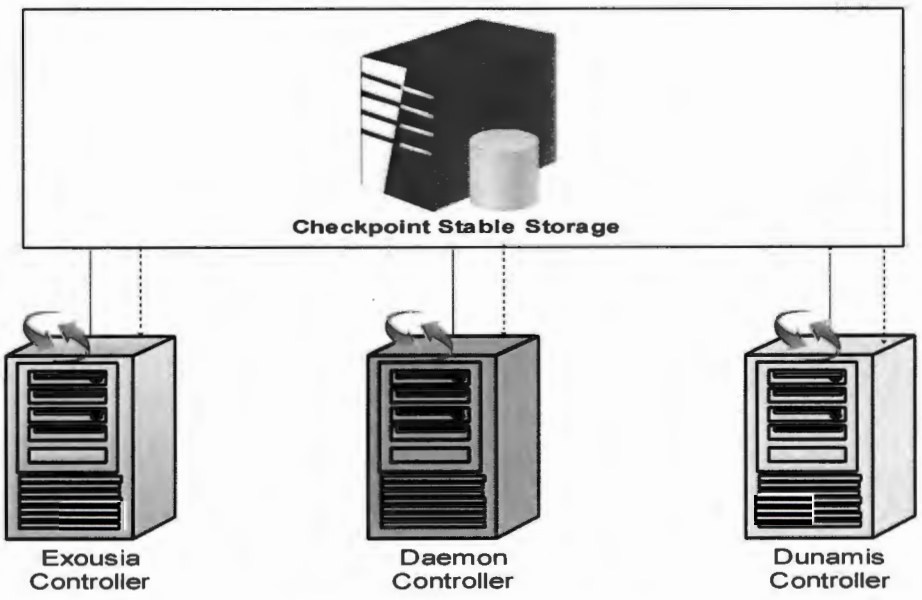


Figure 4.15: Consistency on the EDD Controllers

As indicated on Figure 4.15, the FT Framework allows consistency on the controller by taking checkpoint every 60 seconds and sharing the current state with the backup controller in order to prevent the Exousia controller from being the only controller that has the current state of the network. Since multiple controllers possess the global view of the network by sharing the checkpoint in the stable storage, this makes maintaining controller consistency possible.

4.7 Discussion

In this chapter we introduced a fault tolerance framework which is designed to offer fault tolerance on SDWSN controller. The proposed FT Framework offers continuous network operation during the event of faults arising from the SDWSN controller, by offering advanced fault tolerance strategies on the three dedicated EDD controllers. The proposed EDD controllers have one active controller (Exousia controller) and two standby controllers (Daemon and Dunamis controllers) that have in-built Msg_Mnr each responsible for the communication between the controllers and the Heartbeat manager. (See Figure 4.16)

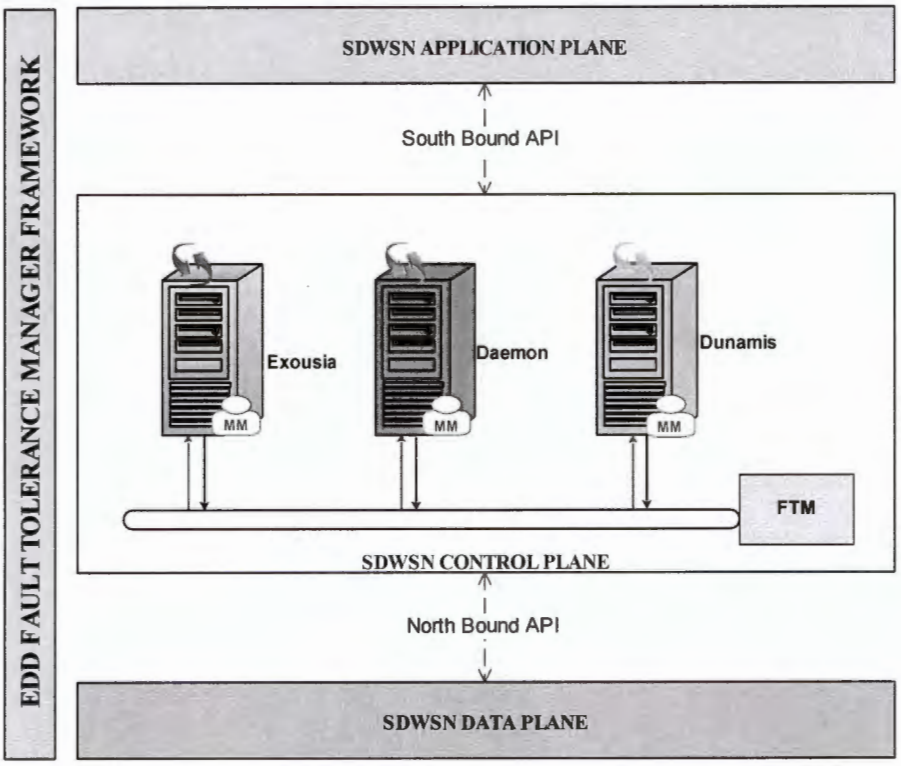


Figure 4.16: Proposed SDWSN architecture [17]

The HBM is used to detect faults that occurred in the network and invoke the FTM to diagnose the cause of failures, plan for recovery and resolve the detected fault immediately. The FTM uses the Voting Manager to select a controller suitable to take over the network. The FTM holds the FT intelligence so it should always be available to cater for FT. The Backup FTM will serve as a standby in case the FTM fails. Both the FTM and its backup resides in the control plane. The control plane ensures the availability of the controllers on network and also enables direct communication protocols between distributed controllers and the other planes. Figure 4.16 presents a summary of the FT framework and a high level view of the entire SDWSN network which is integrated with the FTM.

4.8 Chapter Summary

The purpose of this chapter was to design a FTF for SDWSN controller to ensure the reliability of the SDWSN controller. It is aimed at solving challenges faced by SDWSN as stated in the problem statement. The framework takes into consideration, FT on the controller, consistency on the distributed controllers and scalability. For synchronization, checkpoint techniques are used to capture and monitor the latest state of the network and synchronize it to 3-phase controllers.

CHAPTER 5

Framework Evaluation and Results

5.1 Chapter Outline

This chapter presents the analysis and results of the experiments performed in this research. It begins with an overview of the experiment, its setup and configuration, the process involved and the simulations results. Moreover, we also present a theoretical evaluation of the FT framework for SDWSN proposed in this research.

5.2 Overview

The SDWSN FT framework that was presented in Chapter four introduced FT techniques that will ensure that the controller remains functional in the event of faults on the controller. Thus, this chapter is geared towards evaluating the FT technique proposed in EDD FT framework to validate the framework and its performance. In this case, a theoretical evaluation is used to show the characteristics and the importance of proposed framework. Moreover, a simple simulation is also performed on the SDWSN network using Mininet and OpenDayLight (ODL) controllers to virtualize the network. The simulation aims to demonstrate part of the proof of concept by showing how the ODL controllers connect to the data plane using Mininet.

In particular, the ODL controller is an open source controller that serves as a central intelligence of the entire network, offers network programmability and supports OpenFlow protocols and other SDN standards. On the other hand, Mininet is a powerful network-oriented software application that has the capability to emulate a network and generate a network that supports OpenFlow-based controllers, switches, host, and create a link between these network devices. Thus, the importance of performing simulations in this research is to demonstrate the impact of the controller on the entire SDWSN network. It is then imperative to ensure reliability and availability on the SWDSN controller.

5.3 Simulations

In this section, we discuss the process involved in the simulations performed in this work to evaluate the performance of the proposed FT framework in the SDWSN.

5.3.1 Objective

This research is computer network based, therefore setting up a network simulation is one of the methods that can be used to assess the performance of FT framework for the SDWSN by showing the communication between various network devices. In the context of this research, the position is to show the interactions between components such as the OpenDayLight (ODL) Controller, the OpenFlow switches, the Hosts, and the link between these components or devices. However, the object of this simulation is not to implement features or the operations of the proposed FT framework discussed in Chapter 4, but to demonstrate proof of concept on the simulation of the SDWSN, present performance results and behavior of the network. The implementation of the FT framework is still ongoing due to time and resource constraints.

5.3.2 Setup and Configuration

This section explains the processes or steps that are involved in setting up these experiments, therefore in order to set the environment the following tools are required. The system requirements that are important for the simulations are as follows:

i. Virtual Box (Version 5.1.26 r117224 (Qt5.6.2))
ii. Mininet (Mininet-2.2.2) [105]
iii. OpenDayLight ODL Controllers (Distribution-karaf-0.4.3-Beryllium-SR3)
iv. JAVA programming language
v. Linux Server (Ubuntu-16.04.2-server-amd64)
vi. PuTTY (Release 0.69)

In order to carry out the simulation, the initial step involved running different Linux servers that have different distributed ODL SDN controllers (i.e. Exousia, Daemon and Dunamis) and Mininet. This is then followed by the configuration of all the servers that run the Mininet and ODL controllers to have static IP addresses that allow both server to communicate with each other remotely. In this case, the following static IP addresses were given: Mininet (192.168.56.101), Exousia Controller ODL50 (192.168.56.50), Daemon Controller ODL51 (192.168.56.52) and Dunamis Controller ODL52 (192.168.56.52) respectively.

Note that the word "ODL Controller" will be used interchangeably to denote both controllers (Exousia, Daemon, and Dunamis). Also, with just a single line of command the Mininet allow one to simulate a network by defining the host of the controllers, the network topology alongside the mac address of each OpenFlow device.

The following steps are performed:

A. ODL controllers and Mininet connection

Initially, the Mininet and the defined three ODL Controllers were allocated with static IP addresses which make it possible for Mininet to connect with the controller. To ensure that both are connected and working, Figure 5.1 shows that the ODL Controllers can be accessed by Mininet using PuTTY. Accordingly, the following commands are executed to show that Mininet has access and can communicate to the three ODL Controllers: *ping -c1 192.168.56.50; ping -c1 192.168.56.51; ping -c1 192.168.56.52.* (See Fig. 5.1) Note that Mininet and the Controllers run on different servers but they are configured to communicate with each other.

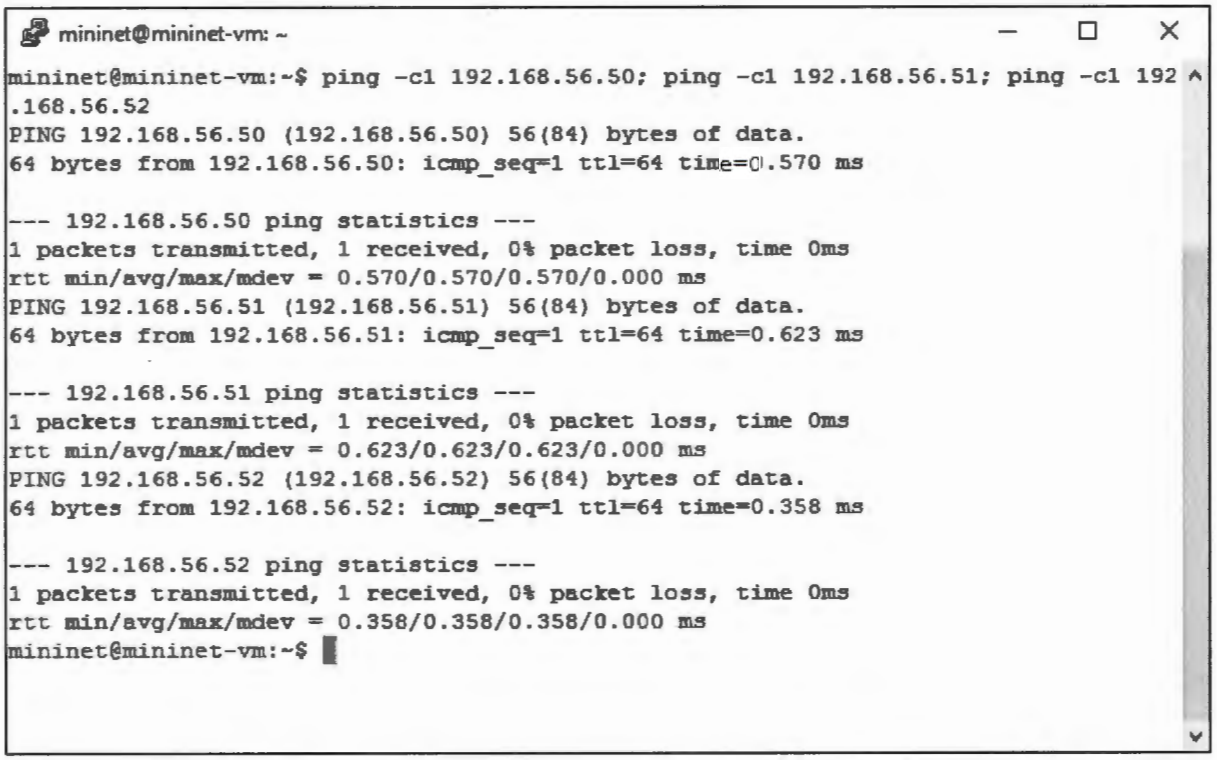


Figure 5. 1: Setting up Distributed ODL Controllers

B. Running ODL controller

After the connection is set, the next step involved running the ODL controller which requires a root privileges to have access. Once the configuration of the ODL controller is successful, before the application is run, one is required to specify the JAVA_HOME directory. To achieve this, the command: *export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64* is used. With the specification of the JAVA_HOME directory, the command: */bin/karaf* is used to start the

ODL Controller by calling the karaf script that resides on the controller bin directory. The command will start the ODL which requires the installation of various features of the ODL such as; odl-restconf, odl-l2switc-switch (ODL layer-2 learning switch), odl-mdsal-apidocs, odl-dlux-core (graphical user interface), odl-dlux-all. Figure 5.2 shows how to run the SDN Exousia Controller. Note, the same process is applicable to the Daemon and the Dunamis ODL Controller before defining the controller on Mininet.

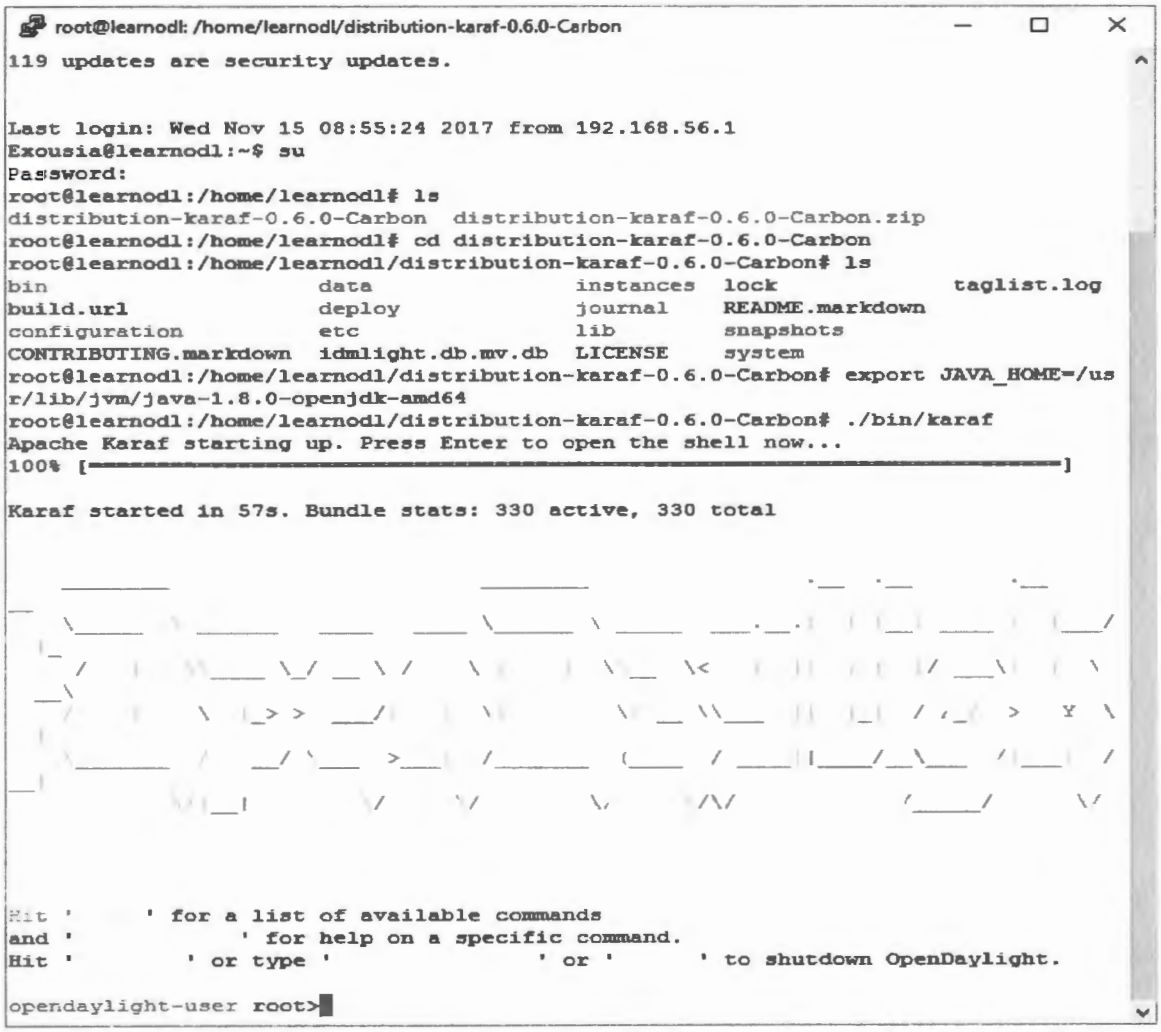


Figure 5. 2: Running ODL Controllers

Running the ODL using command line is very fast and efficient, and requires about 55 seconds to start the controller (see Fig. 5.2). Moreover, the ODL also offers a graphical user interface (GUI) that aids in visualizing the network topology, node information, Yang UI, Yang Visualizer, packets information and other features that can be installed during the configuration since there is no GUI in the Mininet. However, to login to the ODL GUI one needs to log in as indicated on Figure 5.3.



Figure 5. 3: No Devices connected

At this stage, there are no nodes, topology and devices currently connected. Accordingly, when the RELOAD button is clicked, the network topology will remain blank (see Fig. 5.3). The topology can only be visible once configured on the Mininet. The controller can also be accessed using the web browser and assigning the IP address of the controller.

C. Mininet network topology

The Mininet supports various types of topology such as single topology, reversed topology, tree topology, linear topology, torus topology, etc. To simulate the SDWSN on Mininet requires a definition of network topology. There are two scenarios that can be used to create a Mininet topology: the first scenario will create a customized topology that can be executed by using the *mn* command while the second scenario uses customized topology that can be created by writing a python script run from the system shell without using the *mn* command to create the same topology.

In the context of the simulations in this research, we will apply the first scenario which involved using *mn* command. The defined topology is the tree topology which can be created using the following commands:

i. Use *sudo mn -c* command to clear any existing topology or execution on Mininet and allow fresh start.

ii. Execute *sudo mn --controller=remote, ip=192.168.56.50 --mac --topo=tree, depth=3, fanout=5;* command to create out topology on Mininet.

iii. *Connect the other controllers using the following commands respectively; sudo mn --controller=remote, ip=192.168.56.51 --mac --topo=tree, depth=3, fanout=5; sudo mn --controller=remote, ip=192.168.56.52 --mac --topo=tree, depth=3, fanout=5* command to create out topology on Mininet.

To this end the ODL controller is connected remotely using the tree topology with a depth of 3 and a fanout of 5 (see Fig. 5.4). Mininet automatically connects to the specified remote controller, also added are 125 hosts (h1-h125), and 31 switches (s1-s31). Once the creation and addition is done, Mininet then creates a link between switches and host. Lastly, it configures the host, and starts the controller and the switches. By default, when *–mac* is specified, Mininet gives each host and switch different IP addresses.

```
mininet@mininet-vm: ~
*** Adding controller
Connecting to remote controller at 192.168.56.50:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h
52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64 h65 h66 h67 h68 h69 h70 h71 h72 h73 h74 h75 h76
 h77 h78 h79 h80 h81 h82 h83 h84 h85 h86 h87 h88 h89 h90 h91 h92 h93 h94 h95 h96 h97 h98 h99 h100
h101 h102 h103 h104 h105 h106 h107 h108 h109 h110 h111 h112 h113 h114 h115 h116 h117 h118 h119 h12
0 h121 h122 h123 h124 h125
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s22 s23 s24 s25 s26 s27
 s28 s29 s30 s31
*** Adding links:
(s1, s2) (s1, s8) (s1, s14) (s1, s20) (s1, s26) (s2, s3) (s2, s4) (s2, s5) (s2, s6) (s2, s7) (s3,
h1) (s3, h2) (s3, h3) (s3, h4) (s3, h5) (s4, h6) (s4, h7) (s4, h8) (s4, h9) (s4, h10) (s5, h11) (s
5, h12) (s5, h13) (s5, h14) (s5, h15) (s6, h16) (s6, h17) (s6, h18) (s6, h19) (s6, h20) (s7, h21)
(s7, h22) (s7, h23) (s7, h24) (s7, h25) (s8, s9) (s8, s10) (s8, s11) (s8, s12) (s8, s13) (s9, h26)
 (s9, h27) (s9, h28) (s9, h29) (s9, h30) (s10, h31) (s10, h32) (s10, h33) (s10, h34) (s10, h35) (s
11, h36) (s11, h37) (s11, h38) (s11, h39) (s11, h40) (s12, h41) (s12, h42) (s12, h43) (s12, h44) (
s12, h45) (s13, h46) (s13, h47) (s13, h48) (s13, h49) (s13, h50) (s14, s15) (s14, s16) (s14, s17)
(s14, s18) (s14, s19) (s15, h51) (s15, h52) (s15, h53) (s15, h54) (s15, h55) (s16, h56) (s16, h57)
 (s16, h58) (s16, h59) (s16, h60) (s17, h61) (s17, h62) (s17, h63) (s17, h64) (s17, h65) (s18, h66
) (s18, h67) (s18, h68) (s18, h69) (s18, h70) (s19, h71) (s19, h72) (s19, h73) (s19, h74) (s19, h7
5) (s20, s21) (s20, s22) (s20, s23) (s20, s24) (s20, s25) (s21, h76) (s21, h77) (s21, h78) (s21, h
79) (s21, h80) (s22, h81) (s22, h82) (s22, h83) (s22, h84) (s22, h85) (s23, h86) (s23, h87) (s23,
h88) (s23, h89) (s23, h90) (s24, h91) (s24, h92) (s24, h93) (s24, h94) (s24, h95) (s25, h96) (s25,
 h97) (s25, h98) (s25, h99) (s25, h100) (s26, s27) (s26, s28) (s26, s29) (s26, s30) (s26, s31) (s2
7, h101) (s27, h102) (s27, h103) (s27, h104) (s27, h105) (s28, h106) (s28, h107) (s28, h108) (s28,
 h109) (s28, h110) (s29, h111) (s29, h112) (s29, h113) (s29, h114) (s29, h115) (s30, h116) (s30, h
117) (s30, h118) (s30, h119) (s30, h120) (s31, h121) (s31, h122) (s31, h123) (s31, h124) (s31, h12
5)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27
 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h
52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64 h65 h66 h67 h68 h69 h70 h71 h72 h73 h74 h75 h76
 h77 h78 h79 h80 h81 h82 h83 h84 h85 h86 h87 h88 h89 h90 h91 h92 h93 h94 h95 h96 h97 h98 h99 h100
h101 h102 h103 h104 h105 h106 h107 h108 h109 h110 h111 h112 h113 h114 h115 h116 h117 h118 h119 h12
0 h121 h122 h123 h124 h125
*** Starting controller
c0
*** Starting 31 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s22 s23 s24 s25 s26 s27
 s28 s29 s30 s31 ...
```

Figure 5. 4: Setting up a network topology

D. Using ODL GUI to View the network topology

Once the topology is created in Mininet, the topology can then be viewed using the ODL GUI. This is captured in Figure 5.5. In Figure 5.5, a core switch was created and all other switches are connected to it with a depth of 3 and a fanout of 5. Accordingly, before the pingall command is executed, the topology will display leave switches only without showing the host connected to each host.

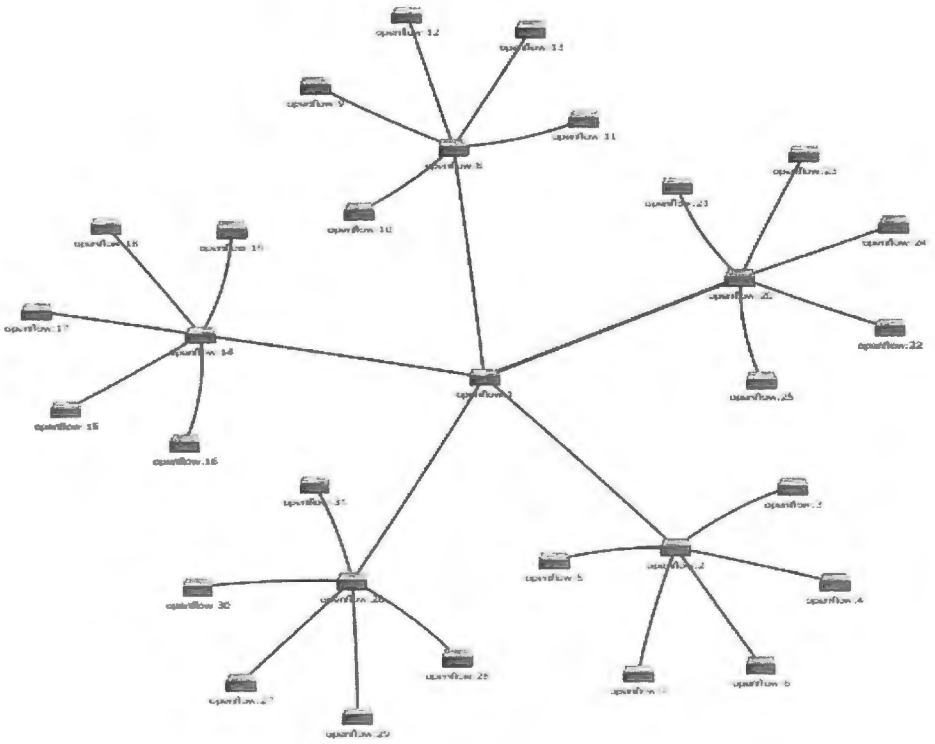


Figure 5. 5: Tree network topology

In the section that follows, the pingall command will be executed to show how the host are connected to the leave switches. The choice of a tree topology is due to its simplicity in command and its effectiveness in the simulation of large networks.

E. Packets received

In order to access the packets received, each leave switch defined on the topology contains a number of hosts connected to it which is specified on the fanout command. In the context of this research, 5 hosts are connected since the specified fanout is 5. No host is connected to the core switch but will only have the leave switch connected to it. During the execution of the *pingall* command, the number of packets successful were 15500/15000, 0 packets were dropped (See Fig. 5.6) consequently, the controllers' power does not reside in the GUI of the

ODL, but in the programmability of the network from the applications that uses the controller as a conduit.

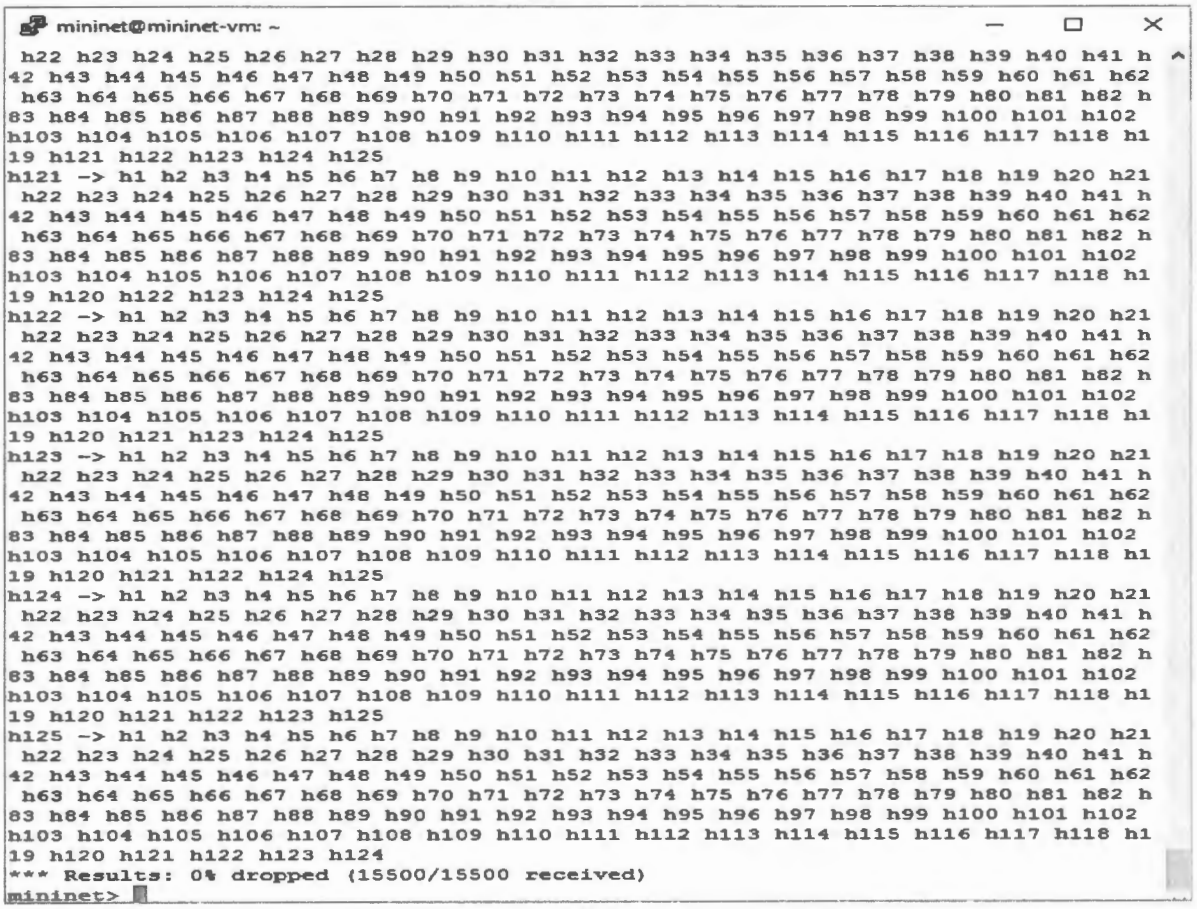


Figure 5. 6: Packets received

F. Hosts and switches connection

To connect the switches to their hosts, *pingall* command is used to help the controller discover all the devices in the topology. In the simulation, by using the topology module on the ODL and executing the *pingall* command, about 125 hosts are observed connected to the leave switch having hosts IP addresses that ranges from 10.0.0.1 to 10.0.0.125 and 31 switches as defined in Mininet (see Fig. 5.7). The OLD GUI clearly show the connection between the hosts and the witches which means that the ODL controller has all the information about the devices that are connected to the network.

The IP addresses on the hosts are defined by including the command --mac when defining the topology. If we don't use the --mac command, every time we execute a pingall command the mac address of each host constantly changes to a random value, therefore using the --mac

command prevents this from happening. On each connection the node information and the port information is shown by putting the cursor on the connector as shown in Figure 5.9 (A).

Note that the Controller has access to all switches on the network such that if one switch can fail the controller can generate alternative parts to forward packets.

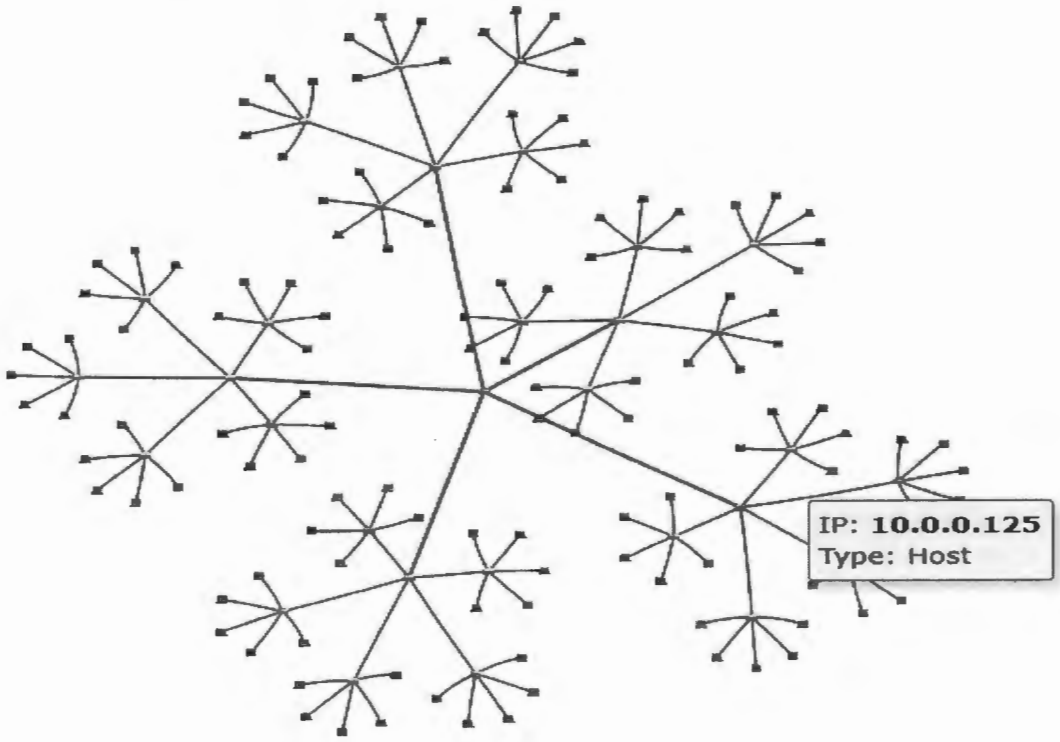


Figure 5. 7: 125 host with defined IP addresses

5.4 Simulations Result

This section presents the results of the simulation performed on the SDWSN using the Mininet network software.

5.4.1 Nodes information

Nodes on the ODL are the switches that are connected to each other. Nodes information can be viewed on the ODL controller as shown in Figure 5.8. The node Id field represent a unique Id of the OpenFlow switch and the node name is the name of the OpenFlow switch. Switches can be given specific names but in the case of this work, no name was defined but identified using the unique Node Id. Furthermore, the node connectors show how many connections each OpenFlow switch has. Accordingly, the statistics column has two different fields: 1.) Flows, which defines the statistics of the flow table for a particular Node Id, 2.) Node Connectors

which shows the node connector statistics for each node Id. In Figure 5.8 each Node Id (Switch) has 7 node connections, except the core switch (OpenFlow: 1) which has 6, because it is connected to 5 switches or nodes as shown in figure 5.8.

Node Id	Node Name	Node Connectors	Statistics
openflow:14		7	Flows | Node Connectors
openflow:13		7	Flows | Node Connectors
openflow:12		7	Flows | Node Connectors
openflow:11		7	Flows | Node Connectors
openflow:10		7	Flows | Node Connectors
openflow:31		7	Flows | Node Connectors
openflow:30		7	Flows | Node Connectors
openflow:8		7	Flows | Node Connectors
openflow:9		7	Flows | Node Connectors
openflow:2		7	Flows | Node Connectors
openflow:3		7	Flows | Node Connectors
openflow:1		6	Flows | Node Connectors
openflow:6		7	Flows | Node Connectors
openflow:7		7	Flows | Node Connectors
openflow:4		7	Flows | Node Connectors

Figure 5. 8: Nodes information

5.4.2 OpenDayLight Network Statistics

OpenDayLight has a module called node, which enables us to observe important network statistics and other information related to the mac address or ports of all switches defined on Mininet. The node module is shown in Figure 5.8.

This section show the result presented by the ODL controller. For example, the Node Id: 12 (switches) is selected. However, the same process is also applicable to other OpenFlow switch. In this case, the analysis will be on both the *Node Connectors and Statistics* (*Node Connectors Stats and Flows Table Stats*). (See Figure 5.9).

A. Node connectors info

A click on the Node Connector number will produce information about Node Id, name port, the mac address, ports number per switch, etc. The Node Id shows the Node Connector Id, the name of the Node connector, the Port Number and the Mac address of each Node Connector Id. For instance, in Figure 5.9, the first *Node connector Id* has an interface of OpenFlow: 12:4, the name allocated is s1-eth4, which connects through port 4 and a mac address of 7e:d8:fe:cd:fb:39 respectively.

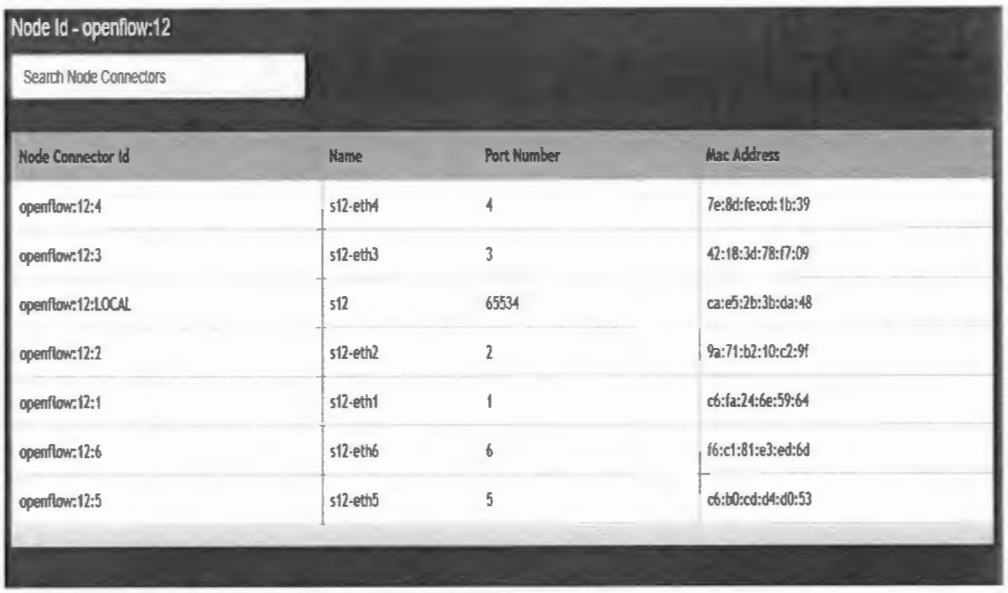


Figure 5. 9: Node Id –OpenFlow :12

B. Node connectors' statistics

In terms of the node statistics shown in Figure 5.10, the OpenFlow Switch: 12 has 7 interfaces which start from interface 1 to interface 6 and a local management interface. Each interface contains 12 statistics columns linked to it. At this stage, we can view the Received Packets (Rx Pkts), Transferred Packets (Tx Pkts), Received Bytes (Rx Bytes), and Transferred Bytes (Tx Bytes), etc.

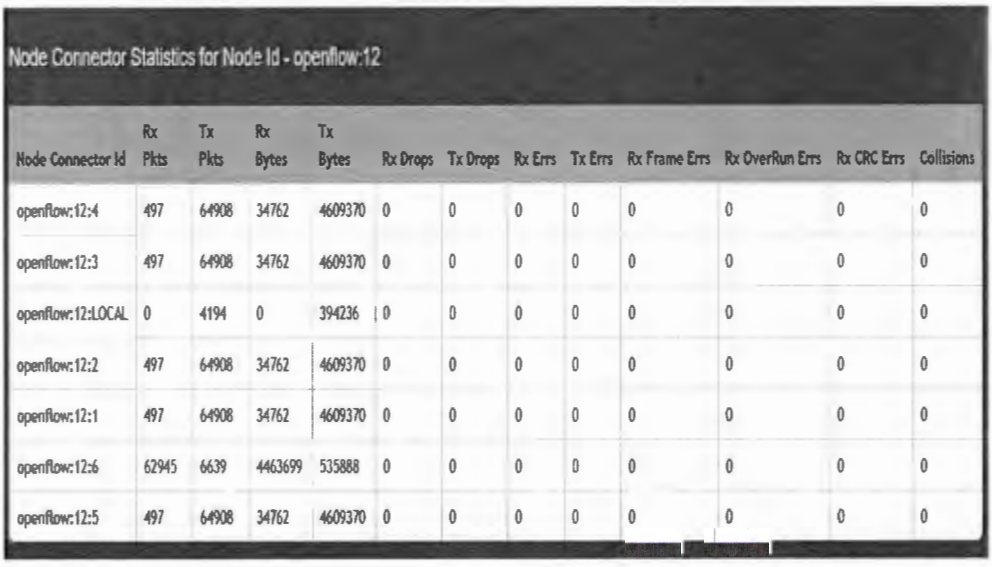


Figure 5. 10: Node Connector statistics for Node Id-OpenFlow: 12

As shown in Figure 5.10, every time the pingall command is executed, the byte counts on each node are always increasing. For example, OpenFlow: 12:4 received 497 (Rx) Pkts, transmitted

64908 (Tx) Pkts, has 34762 Rx Bytes, 4509370 Tx Bytes. With 0 Rx Drops, 0 Tx Drops, 0 Rx Errs, 0 Tx Errs, 0 Rx Frame Errs, 0 Rx OverRun Errs, 0 Rx CRC Errs and 0 Rx Errs.

C. Flow table statistics

On the ODL node module, the flow table statistics associated to a node Id can be viewed by clicking on the flows on the statistics column. As shown in Figure 5.11, the flow table statistics defines the Table Id, active flow, packet match and so on.



Figure 5. 11: flow table statistics for Node Id

Moreover, as an example, Figure 5.12 shows the Active Flow, Pkts Matched, Pkts Locked-up for OpenFlow switch 12. This means that there are 8 Active Flow, 2159776 Pkts Matched, 215977 Pkts that are Locked-up.



Figure 5. 12: Flow Table Stats for Node ID – OpenFlow: 12

5.5 Theoretical Evaluation of SDWSN FT Framework

This section theoretically evaluated the proposed EDD FTM framework discussed in chapter four. Because the proposed framework is not yet implemented, it is therefore important that the framework is theoretically evaluated to ensure that it has a standardized design with respect to fault tolerant system's requirements. The essence is to enhance comprehension for future researchers who may want to implement the idea or add to it. Currently, existing simulation software have limitations for implementing multiple SDWSN controllers which communicate with each other directly in terms of FT strategy execution. Therefore, this lack of FT feature on existing simulators is one factor that has impeded us from using the simulator to validate the proposed FT framework, though it is still ongoing. More time is needed to configure or reprogramme the source code of the controller in order to embed FT functionality before developing the FTM module that sits in the control plane.

In summary, several existing open source simulators required major changes of the entire simulator in order to embed the FT model with fault detection and recovery capabilities. While features of the EDD FTM framework follow the fault and detection cycle presented in section 4.5.2, the evaluation in this chapter will be based on selected features of FT techniques exhibited by the EDD FTM framework as well as selected parameters that are core components of FT strategies used in existing FT techniques as discussed in chapter 3. The evaluation of the EDD FTM framework is captured in Table 5.1.

Table 5.1 presents an evaluation of the FT framework proposed in this research which is based on the following selected parameters: distributed controllers, synchronization, checkpoint manager, voter manager, heartbeat manager, Backup FTM and Msg_Mnr. Table 5.2 shows the evaluation of the fault tolerance requirement on the proposed EDD FTM framework, based on the following criteria: Exactly-once property, Non-blocking, Fault Detection, Fault Recovery, Checkpointing and Overheads (communication and computation).

Table 5. 1: Theoretical evaluation

Criteria	Distributed SDWSN controllers [106], [107] or 3-Phase SDWSN controllers
Description	- The framework introduced multiple or 3-phase controller and FTM that serves as back-up/standby in case of failure and the manager of the FT execution respectively. We designed it in a way that allows the standby controllers to simultaneously have the current state of the network – state consistency in the case of failure. This eliminates single point network failure. - Having the current state of the network by standby controllers enables efficient and speedy fault recovery in the network when failure of the active controller is noticed. - Each controller (E0, D1 and D2) also has a message manager called the Msg_Mnr or MM which is responsible for communicating directly with the HBM in the FTM to perform effective monitoring or detection of fault/failure of the controllers. This is a new feature compared to existing ones.
Component:	This is part of the recovery execution
Criteria:	SDWSN Controller Synchronization [108], [109]
Description:	- The Framework supports synchronization of the controller. - These controllers use checkpointing techniques to have the current state of the network, even controllers that are on standby. - If the active controller fails, the standby controllers can still take over from the failed controller.
Component:	This is part of recovery execution in the FT framework
Criteria:	FTM and Backup FTM [93]
Description:	- The FTM is the manager of the FT strategy like the controller is the manager of the network. - The FT framework is designed to enable Backup FTM to take over from the FTM in the event of failure. Without the Backup FTM, the FTM becomes a single point of failure on the FTM module. - Heartbeat2 in the FTM module is used to exchange heartbeats messages with the Heartbeat2 of the Backup FTM to detect failure on each other.
Component:	This is part of both detection and recovery execution in the FT framework
Criteria:	Checkpoint Manager [110], [111], [112]
Description:	- It is composed of two sub modules that play a critical role for fault recovery - Checkpoint() sub module is responsible for capturing the snapshot of the network state and save it to a stable storage. - During the process of recovery, the Roll_Back () sub module consumes the lasts checkpoint saved from stable storage and restores latest network state to resume network execution. - The latest state or checkpoint of the network is automatically synchronized with the two passive controllers (D1 and D2).
Component:	This is part of the recovery execution
Criteria:	Voting Manager [64]

Description:	- During fault recovery the voting manager is embedded with intelligent mechanism of selecting a controller with the latest state of the network. - In a case of active controller failure, the Voting manager also uses information gathered from the HBM table to check the health of standby controllers before granting it permission to take over the failed controller. - This feature introduces a different voting technique that is new and has not been used in existing voting techniques
Component:	This is part of the fault recovery execution
Criteria:	Heartbeat Manager [113], [114]
Description:	- The HBM is the engine room of FTM that is specifically designed for ultimate fault/failure detection controllers and the Backup FTM. - The FT frame work introduced a new approach where information about the controllers is maintained in a temporary store or table about the controllers and the Backup FTM called the HBM Table. - The HBM Table serves dual purpose: Detection of fault or failure of the controller and Backup FTM and aiding Voting manager to conduct voting to elect a new controller in case of failure using state and health information of the standby controllers.
Component:	This is part of the fault detection execution
Criteria:	Message Manager (Msg_Mnr)
Description:	- The Msg_Mnr module in the controller is a mechanism that enables the HBM to monitor the state of the controller and detect faults/failures. - The Msg_Mnr is also responsible for transferring data requested by HBM. - In this case, if the Msg_Mnr does not respond to the HBM, the heartbeat declares the controller faulty and alerts the voting manager to select another controller immediately. - This component is a new technique of separating the task of network execution from FT execution. Our FTM is not part of the controller there by allowing the controller to carry out its task efficiently.
Component:	This is part of the fault detection execution

Table 5. 2: Description of Operations on FTF

	Criteria	**Description of Operations on FTF**
1	Exactly-once property EOP [82], [84], [115]	- The framework ensures that only one controller can execute at a time. - The voting manager on the FTM supports the exactly-once property to ensure one controller can execute at time. - The voting manager can use the health status of an appropriate controller to elect a new executing controller and the HBM uses appropriate TIMEOUT to detect fault or failure. - The exactly- once property is also applied between the FTM and the backup FTM, where by the backup can only take over when the FTM has failed.
2	Non-blocking property [85], [121]	- In the FTF, the task of network execution continues executing regardless of the failure that occurred in the executing controller.

		- FTF uses checkpointing and synchronisation of the network state to the 3-phase controllers (executing controller and the backup controllers) to eliminate the single point of failure. - Thus, failure of the executing controller would not block the execution of the network task or block other controllers from taking over.
3	Fault Detection [116], [117]	- The EDD FTM framework accommodates the Fault detection requirement in the FTM - Detect faults on the network and ensures an accurate action is invoked to recover. - The HBM is in charge of monitoring and detecting fault or failure on the controller using heartbeat. - HBM pings request the controller to respond to the heartbeat send in order to acknowledge that it is alive. - Controller uses the Msg_Mnr to respond to the request message that was sent by the HBM. - Failure to respond to the heartbeat after a TIMEOUT means that the controller is there failure has occurred and the diagnostic manager will have to analyse the failure and plan for recovery.
4	Fault Recovery [118], [59]	- Recovery component in the FTF is used to restore controller or failed components to its execution or active state. - The Roll_Back () module is used to invoke recovery.
5	Checkpointing [87], [122]	- Takes the state of the network and stores is into the stable storage at regular interval of 60 sec automatically - Checkpoint () module stores the snapshots of the controller for later usage failure recovery. - Checkpoint goes alongside synchronization to ensure network state consistency on the controllers.
6	Overheads (Communication & computation) [119], [120]	- FTF enables efficient communication between the EDD controllers and FTM with no delays during communication or computation. - To offers better computation during the transmission between the FTM and the EDD controllers there is a processing time interval set to determine further action - HBM sends heartbeat at specific intervals to eliminate incessant communication - The FTM is placed outside the controller in the control plane to ensure the controller is not overloaded with responsibilities that could affect its performance. - Stable storage is designed to override its contents in every checkpoint. - Voting manager applies efficient technique to select a new controller in the event of failure without causing any delays due to low memory and other factors that can cause overheads.

5.6 Discussions

Programmable networks require resourceful controllers in order to overcome failure on the control plane by ensuring resilience FT mechanism. The simulation performed utilized 3 SDWSN ODL controllers which correspond to E0, D1 and D2, 31 OpenFlow switches and 125 virtual host, produced by a tree topology of a fanout of 5 and a depth of 3. The results obtained showed that controller can be reliable with consistent global view of the network.

15500 packets were transmitted in less than 10 seconds with no packets lost. Through the use of ODL controller GUI it was possible to see what the topology of the created network looks like (see Fig. 5.7), with the flexibility to see information about the OpenFlow switches and links between the host and the switches. The node connector statistics showed how each OpenFlow switch is connected with other nodes, bytes and pkts that were received and transmitted. The flow table statistics showed the results of the active flows, pkts matched and pkts lockup. With such capability on the controller, we therefore argue that for proof of concept, the results from the experiments shows the practicality of implementing efficient programmable networks.

There is an ongoing improvement on SDWSN controllers that aims to ensure that the control plan operates effectively with error free. The proposed EDD FT framework is positioned to provide the capability of handling FT on realistic large-scale networks. The framework introduces powerful FT techniques that can be used to provide FT on the controller, thereby eliminating single point of failure. The evaluation in Table 5.1 shows that our FT framework has efficient and intelligent techniques that can be implemented to ensure availability and reliability of SDWSN controller in the network.

5.7 Chapter Summary

In this chapter we have discussed, analyzed and presented the results obtained from the simulations conducted. Data received during the simulation shows the efficiency of using Mininet and ODL to create a large network topology that can run a SDWSN with improved network QoS. The results showed how to connect the ODL controllers which are distributed in nature to have a global view of the network. This shows that the controller is the heart or the brain of the SDWSN. However, the only drawback discovered from Mininet is that it lacks the ability to produce GUI. Consequently, we used the ODL GUI to present and view the results. In the chapter that follows, we will discuss the summary, conclusions, and future works of this research.

CHAPTER 6

Summary, Conclusions and Future Works

6.1 Chapter Overview

This chapter presents a brief summary of this research that was conducted and shows how the aim and objectives were achieved. Moreover, it also discusses the conclusions, recommendations and proposed direction for future work.

6.2 General Summary

The summary of what was done in this research is presented in this section. In Chapter 1, we discussed the introduction and the study background of the research. We presented the problem statement, aim and objectives of study. In addition, we outlined the research questions and the methods of investigation utilized. In Chapter 2, we performed an in-depth literature of SDWSN in the perspective of SDN and WSN taking FT, energy consumption; controller availability, reliability, security and many other requirements into consideration [46]. After carefully considering related and previous works, we were persuaded that the SDWSN technical issues or challenges can be fixed or solved, however, the most significant challenge was on: FT on the controller, distributed controller, synchronization of multiple controllers, and so on.

Moreover, in Chapter 3, we conducted a comparative analysis of current FT strategy with the intention to propose an efficient and optimized FT strategy that can be adapted to SDWSN environment. In this case, Chameleon [84] was identified. Accordingly, in Chapter 4, we proposed an effective framework which is made up of two core components: 3-phase controllers or EDD controllers and the FTM which were designed to enable FT on Openflow-based SDWSN controller. The analysis and design of the FTF were performed and discussed in-depth.

In chapter 5, we performed a simple experiment to show the operations of the controller to present the global or abstract view of the network. In the simulation, the EDD controllers were configured and the ODL GUI was used to view the network topology and devices. However, the FT implementation is still ongoing due to its complexity. The results were obtained and reported. A theoretical evaluation was also performed on the FTF to standby for the ongoing implementation. Using the theoretical evaluation helped to cover the objective needed to be answered by this work.

During the course of the research, we identified various FT challenges and advantages of using FT mechanism to ensure FT on the SDWSN controller. To enable the controller fault tolerant, we proposed a SDWSN FTF which is composed of the EDD controllers and the FTM which offers the capability of providing FT on the controller. The FTM is the engine room of the FTF with a powerful backup to protect itself from being a single point of failure. Moreover, the FTM is placed and runs on the control plane to support less resource consumption.

6.3 Conclusions

In this research we have presented a design of a SDWSN FTF in terms of its analysis, design and operations. We also carried out experiments to show the operation of the EDD controllers and its effectiveness. However, its reliability, that is, FT capability of the SDWSN controller is still on going. A theoretical evaluation was also performed to assess the effectiveness of the proposed FTF which we found is effective, reliable and standardized.

Based on the mode of operations of the FTF, we therefore conclude that introducing the FTF to real-world OpenFlow-based SDWSN controller, will go a long way to make the controller to be resilient from failure. By using the EDD controllers and FTM (i.e. the voting, heartbeat, execution, checkpoint and diagnostic) managers, the controller can be prevented from being a single point of failure.

However, the visible limitation was the lack of funds to purchase powerful SDWSN simulation software which led to the usage of open source software that are not advanced and cannot simulate a SDWSN network effectively. Another limitation was the time constraint to program and configure the SDWSN controllers that reflects the 3-phase controller proposed. For this reason, only the ODL controller and Mininet were configured to simulate the controller for proof of concept.

6.4 Recommendation

In this section we present another solution that can be adopted and merged with our proposed solution to increase fault tolerance on SDWSN. Not all our suggestions may be practical to be integrated in the proposed solution. In chapter 2, we discussed challenges related to wireless sensors, and we discovered energy consumption on SDWSN is one of the critical challenges that needs more research attention. We therefore recommend improvement of energy utilization technique to be integrated with the proposed solution in order to make the battery

life of the sensor longer, because even though the FTF offers FT for the controller if the sensors are dead due to poor energy allocation, there is still a need to propose advanced energy allocation mechanism. We also recommend that FT can be improved by introducing a backup of the checkpoint stable storage that synchronizes with the primary checkpoint storage so that if the primary storage breaks, the backup stable storage will take over the checkpoint storage. Moreover, we also recommend the introduction of a security module on the FTF to handle security issues, which is lacking in existing FT approaches.

Furthermore, bandwidth can be an issue when different applications run on different networks. In order to allocate bandwidth efficiently, we recommend the controller to have a built in bandwidth management strategy to ensures that all the processes running will not experience deadlock situation. Also, to have a full back up of the entire network we propose site servers that can act as a disaster recovery in case of an error or disaster on the network. A clone of the entire network must be created with the intelligence to recover the whole system. Lastly, we also recommend the development of a dedicated controller which will be responsible to handle security issues using security mechanisms in the controller.

6.5 Future work

In terms of future work, we intend to implement the proposed FTF, integrate and modify the design of the FTM to handle FT on both the SDWSN controller and the sensors on a real-world network. The goal of redesigning the FTM is to extend FT capability to both. This will also involve proposing the integration of energy utilization or allocation module with the aim of improving energy allocation and consumption management techniques on WSN.

Lack of improved security management strategy will leave both controllers and sensor at risk of being completely down, to a point where intruders can have access to valuable resources [37], [46]. Also in the future, focus will be on introducing security module on the FTF to address security issues on the EDD controller and wireless sensors. We also intend to propose a new FTF that will be based on MA and not only its FT strategy.

However, the implementation of this future work will require time and budget because the FTF requires reconfiguration of the controller and the introduction or improvement of protocols that FTM that offers FT, security and energy allocation on the SDWSN controller and wireless sensor will be based on.

References

[1] A.-C. G. Anadiotis, L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "Towards a software-defined Network Operating System for the IoT," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, 2015, pp. 579–584.

[2] Z. Han and W. Ren, "A novel wireless sensor networks structure based on the SDN," *Int. J. Distrib. Sens. Networks*, vol. 2014, 2014.

[3] S. Kumari, M. K. Khan, and M. Atiquzzaman, "User authentication schemes for wireless sensor networks: A review," *Ad Hoc Networks*, vol. 27, pp. 159–194, 2015.

[4] A. Akbulut, C. Parmaksizoğlu, A. H. Zaim, and G. Yilmaz, "Energy and distance factor based routing protocol for wireless sensor networks using mobile agents," in *Recent Advances in Space Technologies (RAST), 2011 5th International Conference on*, 2011, pp. 399–402.

[5] S. Randhawa, "Research Challenges in Wireless Sensor Network: A State of the Play," *arXiv Preprint . arXiv1404.1469*, 2014.

[6] Z. Zhang, B. Bockelman, D. W. Carder, and T. Tannenbaum, "Lark: An effective approach for software-defined networking in high throughput computing clusters," *Futur. Gener. Comput. Syst.*, vol. 72, pp. 105–117, 2017.

[7] M. B. Aleksander *et al.*, "Implementation technology software-defined networking in Wireless Sensor Networks," in *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2015 IEEE 8th International Conference on*, 2015, vol. 1, pp. 448–452.

[8] T. Luo, H.-P. Tan, and T. Q. S. Quek, "Sensor OpenFlow: Enabling software-defined wireless sensor networks," *IEEE Commun. Lett.*, vol. 16, no. 11, pp. 1896–1899, 2012.

[9] A. De Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," in *Communications (QBSC), 2014 27th Biennial Symposium on*, 2014, pp. 71–75.

[10] 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 Commun. Surv. Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[11] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Comput. Networks*, vol. 72, pp. 74–98, 2014.

[12] T. Miyazaki *et al.*, "A software defined wireless sensor network," in *Computing, Networking and Communications (ICNC), 2014 International Conference on*, 2014, pp. 847–852.

[13] D. Kreutz, F. 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.

[14] E. Kohno, T. Okazaki, M. Takeuchi, T. Ohta, Y. Kakuda, and M. Aida, "Improvement of assurance including security for wireless sensor networks using dispersed data transmission," *J. Comput. Syst. Sci.*, vol. 78, no. 6, pp. 1703–1715, 2012.

[15] A. Akhunzada *et al.*, "Secure and dependable software defined networks," *J. Netw. Comput. Appl.*, vol. 61, pp. 199–221, 2016.

[16] S. Sezer *et al.*, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, 2013.

[17] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements," *IEEE Access*, vol. 5, pp. 1872–1899, 2017.

[18] W. Ejaz, M. Naeem, M. Basharat, A. Anpalagan, and S. Kandeepan, "Efficient Wireless Power Transfer in Software-Defined Wireless Sensor Networks," no. c, pp. 1–12, 2016.

[19] B. T. De Oliveira, L. B. Gabriel, and C. B. Margi, "TinySDN: Enabling multiple controllers for software-defined wireless sensor networks," *IEEE Lat. Am. Trans.*, vol. 13, no. 11, pp. 3690–3696, 2015.

[20] A. Rahmani and R. Mahmud, "Exploitation of openflow in wireless sensor networks," *Comput. Sci. Netw. Technol.*, vol. 1, p. 594–600., 2011.

[21] K. ElDefrawy and T. Kaczmarek, "Byzantine Fault Tolerant Software-Defined Networking (SDN) Controllers," in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, 2016, vol. 2, pp. 208–213.

[22] L. Doherty, J. Simon, and T. Watteyne, "Wireless sensor network challenges and solutions," *Microw. J.*, vol. 55, no. 8, pp. 22–34, 2012.

[23] R. Masoudi and A. Ghaffari, "Software defined networks: A survey," *J. Netw. Comput. Appl.*, vol. 67, pp. 1–25, 2016.

[24] I. Alsmadi and D. Xu, "Security of software defined networks: A survey," *Comput. Secur.*, vol. 53, pp. 79–108, 2015.

[25] S. Salim, M. Javed, and A. H. Akbar, "A mobile agent-based Architecture for Fault tolerance in Wireless sensor Networks," in *Communication Networks and Services Research Conference (CNSR), 2010 Eighth Annual*, 2010, pp. 276–283.

[26] B. a. Kitchenham *et al.*, "Preliminary guidelines for empirical research in software engineering," *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 721–734, 2002.

[27] O. Cheikhrouhou, "Secure group communication in wireless sensor networks: a survey," *J. Netw. Comput. Appl.*, vol. 61, pp. 115–132, 2016.

[28] A. Willig, "Wireless sensor networks: concept, challenges and approaches," *e i Elektrotechnik und Informationstechnik*, vol. 123, no. 6, pp. 224–231, 2006.

[29] H. Karl and A. Willig, *Protocols and architectures for wireless sensor networks*. John Wiley & Sons, 2007.

[30] V. K. Arora, V. Sharma, and M. Sachdeva, "A survey on LEACH and other's routing protocols in wireless sensor network," *Opt. J. Light Electron Opt.*, vol. 127, no. 16, pp. 6590–6600, 2016.

[31] G. Wu, C. Lin, L. Yao, and B. Liu, "A cluster based WSN fault tolerant protocol," *J. Electron.*, vol. 27, no. 1, pp. 43–50, 2010.

[32] A. Sabharwal, P. Schniter, D. Guo, D. W. Bliss, S. Rangarajan, and R. Wichman, ""In-band full-duplex wireless: Challenges and opportunities," *IEEE JSAC Spec. Issue Full-duplex Wirel. Networks*, 2014.

[33] K. Parmar and D. C. Jinwala, "Concealed data aggregation in wireless sensor networks: A comprehensive survey," *Comput. Networks*, vol. 103, pp. 207–227, 2016.

[34] M.-L. Messai and H. Seba, "A survey of key management schemes in multi-phase wireless sensor networks," *Comput. Networks*, vol. 105, pp. 60–74, 2016.

[35] A. Salhieh, J. Weinmann, M. Kochhal, and L. Schwiebert, "Power efficient topologies for wireless sensor networks," in *Parallel Processing, 2001. International Conference on*, 2001, pp. 156–163.

[36] Y. E. Aslan, I. Korpeoglu, and Ö. Ulusoy, "A framework for use of wireless sensor networks in forest fire detection and monitoring," *Comput. Environ. Urban Syst.*, vol. 36, no. 6, pp. 614–625, 2012.

[37] D. G. Padmavathi and M. Shanmugapriya, "A survey of attacks, security mechanisms and challenges in wireless sensor networks," *arXiv Prepr. arXiv0909.0576*, 2009.

[38] L. Wenxing, W. Muqing, and W. Yuewei, "Design of multi-energy-space-based energy-efficient algorithm in novel software-defined wireless sensor networks," *Int. J. Distrib. Sens. Networks*, vol. 13, no. 7, p. 1550147717718113, 2017.

[39] A. Dixit, F. Hao, S. Mukherjee, T. V Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *ACM SIGCOMM Computer Communication Review*, 2013, vol. 43, no. 4, pp. 7–12.

[40] D. Kreutz, F. M. V Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[41] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, and T. Gayraud, "Software-defined networking: Challenges and research opportunities for future internet," *Comput. Networks*, vol. 75, pp. 453–471, 2014.

[42] F. Olivier, G. Carlos, and N. Florent, "Sdn based architecture for clustered wsn," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2015 9th International Conference on*, 2015, pp. 342–347.

[43] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," in *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, 2010, p. 8.

[44] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, 2007, vol. 37, no. 4, pp. 1–12.

[45] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012, pp. 7–12.

[46] W. Li, W. Meng, and L. F. Kwok, "A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures," *J. Netw. Comput. Appl.*, vol. 68, pp. 126–139, 2016.

[47] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, and Z. Zhang, "Enabling security functions with SDN: A feasibility study," *Comput. Networks*, vol. 85, pp. 19–35, 2015.

[48] T. Hu, K. Xue, W. Wei, and W. Jiang, "LENV: A new light-weighted edge network virtualization framework in software-defined wireless networks," in *Wireless Communications & Signal Processing (WCSP), 2015 International Conference on*, 2015, pp. 1–6.

[49] X. Huang, C. Lin, F. Ren, G. Yang, P. D. Ungsunan, and Y. Wang, "Improving the convergence and stability of congestion control algorithm," in *Network Protocols, 2007. ICNP 2007. IEEE International Conference on*, 2007, pp. 206–215.

[50] H. Kim, M. Schlansker, J. R. Santos, J. Tourrilhes, Y. Turner, and N. Feamster, "Coronet: Fault tolerance for software defined networks," in *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, 2012, pp. 1–2.

[51] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software Defined Wireless Network (SDWN): An evolvable architecture for W-PANs," in *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2015 IEEE 1st International Forum on*, 2015, pp. 23–28.

[52] I. T. Haque, S. Member, N. Abu-ghazaleh, and S. Member, "Wireless Software Defined Networking : a Survey and Taxonomy," pp. 1–32, 2016.

[53] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*, 2015, pp. 513–521.

[54] B. Johnson, "Fault-tolerant microprocessor-based systems," *IEEE Micro*, vol. 4, no. 6, pp. 6–21, 1984.

[55] L. Chen *et al.*, "An Efficient Fault Tolerance Path Finding Algorithm for Improving the Robustness of Multichannel Wireless Mesh Networks," vol. 4, pp. 524–525, 2016.

[56] M. Desai and T. Nandagopal, "Coping with Link Failures in Centralized Control Plane Architectures," *Commun. Syst. Networks (COMSNETS), 2010 Second Int. Conf. on*, pp. 1–10, 2010.

[57] F. Longo, S. Distefano, D. Bruneo, and M. Scarpa, "Dependability modeling of software defined networking," *Comput. Networks*, vol. 83, pp. 280–296, 2015.

[58] N. A. Rashid, N. Jailani, and R. Sulaiman, "Security Architecture for Mobile Agent-Based Shari ' ah Compliant e-Auction Marketplace," *Procedia Technol.*, vol. 11, no. Iceei, pp. 510–517, 2013.

[59] A. Moh'd Al Dweik, I. A. S. Ismail, and S. H. Ahmed, "Fault Tolerance Mobile Agent execution System (FTMAS) modeling and performance analysis," in *Information and Communication Systems (ICICS), 2014 5th International Conference on*, 2014, pp. 1–6.

[60] F. Yousuf and Z. Zaman, "A survey of fault tolerance techniques in mobile agents and mobile agent systems," in *Environmental and Computer Science, 2009. ICECS'09. Second International Conference on*, 2009, pp. 454–458.

[61] A. R. Tripathi *et al.*, "Design of the Ajanta system for mobile agent programming," vol. 62, pp. 123–140, 2002.

[62] A. Verma and K. K. Pattanaik, "Mobile agent based train control system for mitigating meet conflict at turnout," *Procedia - Procedia Comput. Sci.*, vol. 32, pp. 317–324, 2014.

[63] S. Srivastava and G. C. Nandi, "Fragmentation based encryption approach for self protected mobile agent," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 26, no. 1, pp. 131–142, 2014.

[64] F. B. Schneider, "Towards fault-tolerant and secure agentry," in *International Workshop on Distributed Algorithms*, 1997, pp. 1–14.

[65] D. Johansen, K. Marzullo, F. B. Schneider, K. Jacobsen, and D. Zagorodnov, "NAP: Practical fault-tolerance for itinerant computations," in *Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference on*, 1999, pp. 180–189.

[66] A. Fedoruk and R. Deters, "Improving fault-tolerance by replicating agents," in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, 2002, pp. 737–744.

[67] S. Pleisch and A. Schiper, "Modeling Fault-Tolerant Mobile Agent Execution as a Sequence of Agreement Problems," no. Srds, 2000.

[68] N. Gude *et al.*, "NOX: towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[69] 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, p. 3.

[70] P. Berde *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.

[71] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 351–362, 2010.

[72] A. Aissioui, A. Ksentini, A. M. Gueroui, and T. Taleb, "Toward Elastic Distributed SDN/NFV Controller for 5G Mobile Cloud Management Systems," *IEEE Access*, vol. 3, pp. 2055–2064, 2015.

[73] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson, "Pratyaastha: an efficient elastic distributed sdn control plane," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 133–138.

[74] A. Dixit, F. Hao, S. Mukherjee, T. V Lakshman, and R. R. Kompella, "ElastiCon; an elastic distributed SDN controller," in *Architectures for Networking and Communications Systems (ANCS), 2014 ACM/IEEE Symposium on*, 2014, pp. 17–27.

[75] F. Olivier, G. Carlos, and N. Florent, "Sdn based architecture for clustered wsn.," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2015 9th International Conference on*, 2015, pp. 342–347.

[76] R. Huang, X. Chu, J. Zhang, and Y. H. Hu, "Energy-efficient monitoring in software defined wireless sensor networks using reinforcement learning: A prototype," *Int. J. Distrib. Sens. Networks*, vol. 11, no. 10, p. 360428, 2015.

[77] A. Sehgal, "Using the contiki cooja simulator," *Comput. Sci. Jacobs Univ. Bremen Campus Ring*, vol. 1, p. 28759, 2013.

[78] M. Jacobsson and C. Orfanidis, "Using software-defined networking principles for wireless sensor networks," in *11th Swedish National Computer Networking Workshop (SNCNW 2015) Karlstad, May 28-29, 2015*, 2015.

[79] S. Tomovic, M. Radonjic, M. Pejanovic-Djurisic, and I. Radusinovic, "software-defined wireless sensor networks: opportunities and challenges."

[80] S. Pleisch and A. Schiper, "Fatomas-a fault-tolerant mobile agent system based on the agent-dependent approach," in *Dependable Systems and Networks, 2001. DSN 2001. International Conference on*, 2001, pp. 215–224.

[81] D. Johansen, K. Marzullo, F. B. Schneider, K. Jacobsen, and D. Zagorodnov, "NAP: Practical fault-tolerance for itinerant computations.," in *Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference on*, 1999, pp. 180–189.

[82] T. Osman, W. Wagealla, and A. Bargiela, "An approach to rollback recovery of collaborating mobile agents," *IEEE Trans. Syst. Man, Cybern. Part C (Applications Rev.*, vol. 34, no. 1, pp. 48–57, 2004.

[83] P. Arora, S. Vashist, R. Singh, and R. Hans, "Analysis of fault tolerant techniques in secure mobile agent Paradigm," *J. Emerg. Technol. web Intell.*, vol. 6, no. 2, pp. 231–237, 2014.

[84] K. Rothermel and M. Straßer, "A fault-tolerant protocol for providing the exactly-once property of mobile agents," in *Reliable Distributed Systems, 1998. Proceedings. Seventeenth IEEE Symposium on*, 1998, pp. 100–108.

[85] R. Guerraoui, "Revisiting the relationship between non-blocking atomic commitment and consensus," in *Proceedings of the 9th International Workshop on Distributed Algorithms (WDAG-9),* 1995, pp. 87–100.

[86] K. Damasceno, N. Cacho, A. Garcia, A. Romanovsky, and C. Lucena, "Context-aware exception handling in mobile agent systems: the MoCA case," in *Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems*, 2006, pp. 37–44.

[87] R. Singh and M. Dave, "Antecedence graph approach to checkpointing for fault tolerance in mobile agent systems," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 247–258, 2013.

[88] M. M. Khokhar, A. Nadeem, and O. M. Paracha, "An Antecedence Graph Approach for Fault Tolerance in a Multi-Agent," in *Mobile Data Management, 2006. MDM 2006. 7th International Conference on*, 2006, p. 137.

[89] R. K. Iyer, Z. Kalbarczyk, and S. Bagchi, "Chameleon: adaptive fault tolerance using reliable, mobile agents," in *Reliable Distributed Systems, 1997. Proceedings., The Sixteenth Symposium on*, 1997, pp. 61–62.

[90] R. Hans and R. Kaur, "Novel dynamic shadow approach for fault tolerance in mobile agent systems," in *Signal Processing and Communication Systems (ICSPCS), 2012 6th International Conference on*, 2012, pp. 1–6.

[91] C. Chowdhury and S. Neogy, "Checkpointing using mobile agents for mobile computing system," in *Proc. of International Journal of Recent Trends in Engineering*, 2009, vol. 1, no. 2.

[92] R. Singh and M. Dave, "Using host criticalities for fault tolerance in mobile agent systems," in *Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on*, 2012, pp. 67–72.

[93] A. V Sutagundar, V. S. Bennur, and K. N. A. A. M. Bhanu, "Agent based fault tolerance in wireless sensor networks," in *Inventive Computation Technologies (ICICT), International Conference on*, 2016, vol. 1, pp. 1–6.

[94] K. K. Leung and K. W. Ng, "A fault-tolerance mechanism for mobile agent systems," in *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, 2005, vol. 1, pp. 1006–1012.

[95] M. R. Lyu, X. Chen, and T. Y. Wong, "Design and evaluation of a fault-tolerant mobile-agent system," *IEEE Intell. Syst.*, vol. 19, no. 5, pp. 32–38, 2004.

[96] L. M. Silva, V. Batista, and J. G. Silva, "Fault-tolerant execution of mobile agents," in *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*, 2000, pp. 135–143.

[97] L. M. Silva *et al.*, "JAMES: A platform of mobile agents for the management of telecommunication networks," in *International Workshop on Intelligent Agents for Telecommunication Applications*, 1999, pp. 76–95.

[98] R. Guerraoui and A. Schiper, "Software-based replication for fault tolerance," *Computer (Long. Beach. Calif).*, vol. 30, no. 4, pp. 68–74, 1997.

[99] E. Gendelman, L. F. Bic, and M. B. Dillencourt, "An application-transparent, platform-independent approach to rollback-recovery for mobile agent systems," in *Distributed Computing Systems, 2000. Proceedings. 20th International Conference on*, 2000, pp. 564–571.

[100] Y. Huang and C. Kintala, "Software implemented fault tolerance: Technologies and experience," in *FTCS*, 1993, vol. 23, pp. 2–9.

[101] P. Chhabra and H. L. Mandoria, "Exploration of Various Fault Tolerance Techniques," *Int. J. Comput. Appl.*, vol. 162, no. 11, 2017.

[102] Y. Lu *et al.*, "A study on the reliability of software defined wireless sensor network," in *Smart City/SocialCom/SustainCom (SmartCity), 2015 IEEE International Conference on*, 2015, pp. 129–134.

[103] B. Isong and E. Bekele, "Fault Tolerance in Mobile Agents: State-of-the-Art and Challenges." 2009.

[104] R. Huang, X. Chu, J. Zhang, and Y. H. Hu, "Scale-free topology optimization for software-defined wireless sensor networks: A cyber-physical system," *Int. J. Distrib. Sens. Networks*, vol. 13, no. 6, p. 1550147717713626, 2017.

[105] R. L. S. De Oliveira, A. A. Shinoda, C. M. Schweitzer, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, 2014, pp. 1–6.

[106] A. Aissioui, A. Ksentini, and A. Gueroui, "An efficient elastic distributed SDN controller for follow-me cloud," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on*, 2015, pp. 876–881.

[107] M. Abolhasan, J. Lipman, W. Ni, and B. Hagelstein, "Software-defined wireless networking: centralized, distributed, or hybrid?," *IEEE Netw.*, vol. 29, no. 4, pp. 32–38, 2015.

[108] T. Mizrahi and Y. Moses, "ReversePTP: A software defined networking approach to clock synchronization," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 203–204.

[109] L. Schiff, S. Schmid, and P. Kuznetsov, "In-band synchronization for distributed sdn control planes," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 1, pp. 37–43, 2016.

[110] P. Arora and R. Singh, "Improved checkpoint mechanism in mobile agent systems for information retrieval applications," in *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*, 2015, pp. 667–671.

[111] R. Singh and M. Dave, "Antecedence graph approach to checkpointing for fault tolerance in mobile agent systems.," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 247–258, 2013.

[112] J. Cao, G. H. Chan, and T. S. Dillon, "Checkpointing and rollback of wide-area distributed applications using mobile agents," in *Parallel and Distributed Processing Symposium., Proceedings 15th International*, 2001, p. 6 pp.

[113] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: high-bandwidth multicast in cooperative environments," in *ACM SIGOPS Operating Systems Review*, 2003, vol. 37, no. 5, pp. 298–313.

[114] Y. Yu, K. Li, W. Zhou, and P. Li, "Trust mechanisms in wireless sensor networks: Attack analysis and countermeasures," *J. Netw. Comput. Appl.*, vol. 35, no. 3, pp. 867–880, 2012.

[115] L. Zeghache and N. Badache, "Optimistic replication approach for transactional mobile agent fault tolerance," in *Software Engineering Artificial Intelligence Networking and Parallel/Distributed Computing (SNPD), 2010 11th ACIS International Conference on*,

2010, pp. 193–198.

[116] A. S. da Silva, P. Smith, A. Mauthe, and A. Schaeffer-Filho, "Resilience support in software-defined networking: A survey," *Comput. Networks*, vol. 92, pp. 189–207, 2015.

[117] S. E. George, I.-R. Chen, and Y. Jin, "Movement-based checkpointing and logging for recovery in mobile computing systems," in *Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access*, 2006, pp. 51–58.

[118] P. Patil, P. Zavarsky, D. Lindskog, and R. Ruhl, "Fault Tree Analysis of Accidental Insider Security Events," in *Cyber Security (CyberSecurity), 2012 International Conference on*, 2012, pp. 113–118.

[119] D. Goyal and M. R. Tripathy, "Routing protocols in wireless sensor networks: a survey," in *Advanced Computing & Communication Technologies (ACCT), 2012 Second International Conference on*, 2012, pp. 474–480.

[120] W. Qu, M. Kitsuregawa, H. Shen, and Z. Shan, "A novel fault-tolerant execution model by using of mobile agents," *J. Netw. Comput. Appl.*, vol. 32, no. 2, pp. 423–432, 2009.

[121] H. Hamidi, "A New Approach for Mobile Agent Fault-Tolerance and Reliability Using Distributed Systems," pp. 0–4, 2005.

[122] H. Kim, H. Y. Yeom, T. Park, and H. Park, "The cost of checkpointing, logging and recovery for the Mobile Agent Systems," in *Dependable Computing, 2002. Proceedings. 2002 Pacific Rim International Symposium on*, 2002, pp. 45–48.