

**AN EFFICIENT DISTRIBUTED CONTROL SYSTEM FOR SOFTWARE-DEFINED
WIRELESS SENSOR NETWORKS**

by

Hlabishi Isaac Kobo

Submitted in partial fulfillment of the requirements for the degree
Philosophiae Doctor (Electronics)

in the

Department of Electrical, Electronic and Computer Engineering
Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

June 2018

SUMMARY

AN EFFICIENT DISTRIBUTED CONTROL SYSTEM FOR SOFTWARE-DEFINED WIRELESS SENSOR NETWORKS

by

Hlabishi Isaac Kobo

Supervisor: Dr G.P. Hancke
Co-Supervisor: Dr A.M. Abu-Mafhouz
Department: Electrical, Electronic and Computer Engineering
University: University of Pretoria
Degree: Philosophiae Doctor (Electronics)
Keywords: Industrial Wireless Networks, Software Defined Wireless Sensor Networks (SDWSN), Internet of Things (IoT,), Wireless Sensor Networks (WSN), Software Defined Networking (SDN), SDWSN controller, Fragmentation, Data Consistency Models

The advent of IoT as the next dominant player in cyber circles has ignited much research interest and a closer synergy between communication and computing models. Software-defined networking (SDN) and wireless sensor networks (WSNs) are some of the models envisaged to play a vital role in the IoT framework. SDN is an emerging network paradigm which has disrupted the status quo in networking and computing. This model is currently receiving much research attention and is being adopted rapidly by industry. It introduces flexibility, innovation, simplicity, and better management to networking. On the other hand, WSNs have always been used for monitoring physical and environmental factors such as temperature, humidity, vibrations, motions, seismic events, etc. The introduction and development of smart sensors have improved and advanced the WSNs. The emergence of the Internet of Things (IoT) paradigm has extended the scope of the demand of WSNs as they are considered to be the main building blocks of the IoT. However, WSNs continue to be plagued by challenges such as limited energy, computational capability, data storage, and communication bandwidth. The application of

SDN to WSNs address most of the inherent WSNs challenges which have resulted in a new model of software-defined wireless sensor networks (SDWSNs). The SDWSN model is currently receiving much research attention as it has enormous potential for the future ICT. The SDN model advocates the separation of control logic and forwarding from the network elements. This decoupling leaves the element as a dumb device and centralises the control logic in a controller.

The controller in SDWSN is very vital and critical as it holds the intelligence and control of the whole network. The current major challenge is the centralisation of this controller. This makes the network vulnerable to malicious attacks as it becomes a simple target for adversaries. Another challenge is the fact that it stifles growth as it limits the scalability of the network and could potentially suffer performance degradation. Therefore, the reliability, performance, and efficiency of the network depends on the controller, despite operation. This study proposes an efficient distribution method for the SDWSN control system using the concept of fragmentation. This entails dedicating segments of the network to local controllers; these controllers are small and inexpensive but efficient. It also involves a global controller which has a global view of the network. This two-level architecture will leverage distribution, which will ensure availability and performance enabled by access. The purpose of this research study is to investigate if distributing an SDWSN control system is ideal, as well as to investigate the viability of the fragmentation model to achieve scalability, reliability and better performance. The evaluation shows that distributing the control system of the SDWSN is not only ideal but necessary. The fragmentation model also proved to bring a positive impact on the SDWSN control.

The fragmentation model is envisaged to enhance the participation of SDWSNs in IoT. Therefore, the model is further optimised for ease of integration and deployment efficiency. This entails controller placement and controller re-election after controller failure mechanisms. The controller placement ensures a procedural and structured controller placement which aims at reducing the propagation latency between the sensor nodes and the local controllers as well as between the local controllers and the global controller. The controller re-election ensure that distance is taken into consideration when a controller is replaced by its peer after failure, thus ensures that the chosen replacement is closer to the failed controller. The two mechanisms were evaluated and proven to be efficient and improved performance.

DEDICATIONS

This thesis is dedicated to my late father Malepanta Ezekiel Kobo, Mothailana 'a Ngwato le Boreteadi, whom the Lord called home a few months before the completion of this work. Tubatse, the journey that we started together, you never completed. Thank you, Papa, for the words of wisdom, advice, and prayers. I shall forever treasure your teachings of love, respect, humility, and faith. Robala ka khutšo Mothailana 'a Ngwato, TUBATSE.

ACKNOWLEDGEMENTS

I thank the Almighty God, the Lord Jesus Christ for affording me the ability, wisdom, strength, and fortitude to complete this work.

I would like to thank my supervisors, Dr Gerhard Hancke and Dr Adnan Abu-Mafhouz for the guidance, advice, and direction that they have given me in the course of this research study. Special thanks to Adnan for his immense dedication, enthusiasm, and patience he has shown to me. He went beyond his call of duty and became a friend and a counsellor; I owe him a great deal of gratitude. I would also like to thank Lusani Mamushiane for her valuable contribution to this work, ndo livhuwa nga maanda khaladzi. I also thank the CSIR, Meraka Institute for the financial support. Many thanks to my colleagues whose support and inputs have been invaluable.

I thank my family for the support they have shown me over the years, I love you guys. To my mother Thabitha, ke a leboga Mahlako 'a Pheladi, a Modimo a le šegofatše. When the going was tough, I knew only your call would suffice. Special thanks to my brother Tsimishi for the special support, thanks warra. To my partner Phumi, thank you very much; this road would have been difficult, dull, and lonely without you. Thanks for the love, patience, and encouragement; I love you. Ke a leboga Nareadi 'a Mahlako, ngwetši 'a Mahlako.

I would also like to thank my family in Christ for the support and prayers. Thanks to Pastor Thobejane, Ngwato 'a Bauba, for the advice, encouragement and prayers, Ke a leboga mmina Noko, Modimo a le dire ka go loka.

Gosebo 'a Mothailana

LIST OF ABBREVIATIONS

5G	5 th generation wireless technology
6LoWPAN	IPv6 addressing for low-rate personal networks
ACID	Atomicity, consistency, isolation, and durability
ACL	Access control list
AMQP	Advanced message-queuing protocol
AODV	Ad hoc on-demand distance vector
API	Application programming interface
BASE	Basically available, soft state, eventually consistent
BLE	Bluetooth low energy
BS	Base station
CAN	Content addressable network
CC	Cloud computing
CE	Controlling element
CPU	Central processing unit
CRUD	Create, read, update and delete
CTP	Collection tree protocol
DC	Data centres
DDS	Data distribution service
DoS	Denial of service
FE	Forwarding element
FMC	Follow me cloud
FMCC	Follow me cloud controller
ForCes	Forwarding and control element separation
G-FMCC	Global follow-me cloud controller
Ha	Alternative hypothesis
HaaS	Hardware as a service
Ho	Null hypothesis
IaaS	Infrastructure as a service
ICT	Information and communication technology
IEEE	Institute of Electrical and Electronics Engineers

IETF	Internet Engineering Task Force
IIoT	Industrial Internet of Things
ILP	Integer linear programming
IoT	Internet of Things
IP	Internet protocol
IPv4	Internet protocol version 4
IPv6	Internet protocol version 6
ISA	International Society of Automation
ISM	Industrial, scientific and medical
LFB	Logical function block
L-FMCC	Local follow-me cloud controller
LNMP	LoWPAN network management protocol
LR-WPAN	Low-Rate wireless personal network
LTE	Long-term evolution
MAC	Medium access control
MCC	Mobile cloud computing
MDSE	Model-driven software engineering
MEC	Mobile edge computing
MEMS	Micro-electrical Mechanical Systems
NB	Northbound
NFV	Network functions virtualization
NoS	Network operating system
NOX/S	Network operating system
ODL	OpenDaylight
ONF	Open Network Foundation
ONOS	Open network operating system
OO	Object oriented
OTAP	Over-the-air programming
PaaS	Platform as a service
PER	Packet error rate
PHY	Physical (layer)
RF	Radio frequency
RSSI	Received signal strength indicator

RSU	Road-site-unit
RSUC	Road-site-unit controller
RTT	Round trip time
Rx	Receive
SaaS	Software as a service
SB	Southbound
SD	Standard deviation
SDCSN	Software-defined cluster sensor network
SDN	Software-defined networking
SDNCH	SDN cluster head
SDN-WISE	SDN-wireless sensor network
SDWN	Software-defined Wireless Network
SDWSN	Software-defined Wireless Sensor Networks
SNMP	Simple network management protocol
SOA	Service-oriented architecture
SSL	Secure sockets layer
TCAM	Ternary content-addressable memory
TCP/IP	Transmission control protocol/internet protocol
TLS	Transport layer security
Tx	Transmit
UWB	Ultra-wideband
VANET	Vehicular ad hoc network
VM	Virtual machine
WAN	Wide area networks
WLAN	Wireless local area network
WSDL	Web service definition language
WSN	Wireless sensor networks
XML	Extensible Markup Language
XSD	XML schema definition

TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	1
1.1	BACKGROUND	1
1.2	PROBLEM STATEMENT	2
1.2.1	Context of the problem	2
1.2.2	Research gap	3
1.3	RESEARCH OBJECTIVES AND QUESTIONS	4
1.4	HYPOTHESIS AND APPROACH	5
1.5	RESEARCH GOALS	5
1.6	RESEARCH CONTRIBUTION	6
1.7	RESEARCH OUTPUTS	7
1.8	DELINEATION AND LIMITATIONS	7
1.9	OVERVIEW OF STUDY	7
CHAPTER 2	LITERATURE STUDY	9
2.1	OVERVIEW OF SOFTWARE-DEFINED NETWORKS	9
2.1.1	Architecture	10
2.1.2	Protocols and Standards	11
2.1.3	Applications	13
2.1.4	Security	14
2.2	WSN CHALLENGES	15
2.2.1	Energy	16
2.2.2	Communication	17
2.2.3	Routing	19
2.2.4	Security	20
2.2.5	Configuration	21

2.3	IMPORTANCE OF SDN IN WSN	22
2.3.1	Energy	22
2.3.2	Network Management and Configuration	22
2.3.3	Scalability	23
2.3.4	Routing, Mobility and Localisation	23
2.3.5	Interoperability	24
2.3.6	Communication	24
2.3.7	Security	24
2.3.8	Summary of discussion	25
2.4	RELATED CONCEPTS	25
2.4.1	The application of SDN in the related concepts	25
2.4.2	Summary of discussion	29
2.5	SOFTWARE-DEFINED WIRELESS SENSOR NETWORKS	29
2.5.1	SDWSN Architectures	29
2.5.2	Routing	35
2.5.3	Network Management	39
2.5.4	Summary of discussion	41
2.6	FUTURE RESEARCH CHALLENGES	42
2.6.1	WSN-inherent challenges	42
2.6.2	Network operating system	43
2.6.3	Practical implementation and evaluation	43
2.6.4	Inter and intraplane communication	44
2.6.5	Standardisation	44
2.6.6	Security	44
2.6.7	Distributed-control system	46
2.7	DESIGN REQUIREMENTS	47
2.7.1	Duty cycles	48
2.7.2	In-network data aggregation and decision fusion	48
2.7.3	Flexible definition of rules	48
2.7.4	Node Mobility	48
2.7.5	Unreliability of wireless links	49
2.7.6	Self-healing ability	49
2.7.7	Backward and peer compatibility	49

2.7.8	Data-centric and address-centric (multiple identification)	49
2.7.9	Scalability	50
2.7.10	East/Westbound interface	50
2.7.11	Northbound and Southbound interfaces	50
2.7.12	Security	51
2.8	LESSONS LEARNT	51
2.9	CONCLUSION	53
CHAPTER 3	SOFTWARE-DEFINED WIRELESS SENSOR NETWORK CONTROL SYSTEM	54
3.1	SDN CONTROLLER IMPLEMENTATIONS	54
3.1.1	A single centralised controller	54
3.1.2	Distributed controllers	55
3.1.3	Logically centralised but physically distributed	56
3.1.4	A data plane extension control	57
3.1.5	Switch to controller mapping configurations	59
3.1.6	Summary of discussion	59
3.2	SDWSN CONTROLLER	60
3.3	CONCLUSION	62
CHAPTER 4	FRAGMENTATION-BASED DISTRIBUTED CONTROL SYSTEM FOR SOFTWARE-DEFINED WIRELESS SENSOR NETWORKS	63
4.1	DISTRIBUTION	63
4.1.1	Background	64
4.1.2	Fragmentation	66
4.2	EPIDEMIC/GOSSIP PROTOCOLS	69
4.2.1	Optimisation	74
4.2.2	Time complexity	78
4.3	CONCLUSION	79
CHAPTER 5	RESULTS AND DISCUSSION	80
5.1	EXPERIMENTAL EVALUATION	80
5.1.1	Tools	80
5.1.2	Experimental Setup	81

5.1.3	Evaluation Methods and Procedures	82
5.1.4	Simulation	84
5.2	RESULTS AND DISCUSSION	89
5.2.1	Controller setup time	89
5.2.2	Number of packets	91
5.2.3	Round trip time (RTT)	93
5.2.4	Standard deviation	96
5.2.5	Packet error rate	97
5.2.6	Time variations	98
5.3	CHALLENGES	102
CHAPTER 6	EFFICIENT CONTROLLER PLACEMENT AND MASTER NODE	
	RE-ELECTION	103
6.1	THE CONTROLLER PLACEMENT IN SDWSN	103
6.1.1	Brief Background	104
6.1.2	Brief Literature	104
6.1.3	Controller placement in SDWSN	106
6.1.4	Proposed controller placement	107
6.1.5	Experiment	108
6.1.6	Results and Discussion	112
6.1.7	Latency	120
6.2	CONTROLLER RE-ELECTION PROBLEM	120
6.2.1	Background	120
6.2.2	ONOS device mastership	121
6.2.3	Proposed controller re-election	122
6.2.4	Experiment	124
6.2.5	Results and Discussion	127
6.3	CONCLUSION	129
CHAPTER 7	CONCLUSION	130
7.1	CONCLUSION	130
7.1.1	SUMMARY OF CONTRIBUTIONS	133
7.2	FUTURE RESEARCH WORK	135

REFERENCES 136

LIST OF TABLES

2.1	Common SDN protocols [1]	12
2.2	IEEE 802.15.4 specification [1,2].	18
2.3	WSN standards based on IEEE 802.15.4 [1]	19
2.4	Current SDWSN architectures [1].	35
2.5	Current SDWSN-routing protocols [1]	39
2.6	Security threats on both WSN layers and SDN planes [1]	46
3.1	Distributed controllers in SDN [1].	60
3.2	Detailed comparison of the controllers [1]	61
5.1	The results of the experiments	90
6.1	Sink nodes with coordinates	110
6.2	Coordinates and weights/distances	111
6.3	The locations of the local controllers	112
6.4	The locations of the local controllers after the moving median function	116
6.5	The distances between obtained locations and the centroids	119
6.6	The local controllers and their location coordinates in latitudes and longitudes	126
6.7	The local controllers with their replacements	127

LIST OF FIGURES

2.1	The basic SDN framework with the three planes and a central controller [1].	11
2.2	Basic packet-forwarding flow in OpenFlow [1,3].	13
2.3	Current and future network computing technologies with SDN [1].	28
2.4	Basic SDWSN architecture as currently applied by various studies. Different functionalities are distributed along the three planes [1].	30
2.5	The context-aware and policy based routing model [1,4].	36
2.6	SDN cluster-based routing [1,5].	38
2.7	The hybrid routing model architecture [1,6].	39
2.8	The OpenRoads architecture [1,7].	41
2.9	SDWSN design requirements. This figure captures the requirements as currently applied and also those that should be considered in future [1].	47
3.1	SDN central controller [1].	55
3.2	SDN distributed controllers [1].	56
3.3	SDN logically centralised but physically distributed controllers [1].	57
3.4	SDN data plane extension controller [1].	58
4.1	Distributed-control system for SDWSN with fragmentation [8].	67
4.2	The complete research structure [8].	70
4.3	The flowchart of the Best Effort algorithm with fragmentation [8].	76
4.4	The flowchart of the Anti-entropy algorithm with fragmentation [8].	78
5.1	The experimental design and setup.	83
5.2	The identity of the controller with connected devices.	84
5.3	The flows inside the controller during evaluation.	85
5.4	The node structure under test.	86

5.5	The simulation-timing script.	86
5.6	The simulation control windows.	87
5.7	The exchange of data amongst the motes.	87
5.8	The radio messages of the motes.	88
5.9	The duty cycles of the motes.	88
5.10	The test simulation.	89
5.11	Controller setup times of experiments <i>A</i> , <i>B</i> , and <i>C</i>	91
5.12	Number of packets produced.	92
5.13	Average RTT for 24 nodes.	94
5.14	Average RTT for 30 nodes.	94
5.15	Average RTT for 39 nodes.	95
5.16	Average RTT vs number of nodes.	95
5.17	Average Standard deviation.	96
5.18	The Packet Error rate.	98
5.19	The controller setup for the extended experiments.	99
5.20	Average RTT for the extended experiments with 39 nodes for one hour.	99
5.21	Average RTT for the extended experiments with 39 nodes for five hours.	100
5.22	The standard deviation for the extended experiments.	100
5.23	The packet error rate for the extended experiments.	101
6.1	The locations of the local controllers by k-means algorithm.	112
6.2	The locations of the global controllers by k-means algorithm.	113
6.3	The use of the median is the same as that of the mean.	114
6.4	Large topology does not change the placement of the global controller when using k-means.	115
6.5	The locations of the local controller after re-optimisation.	117
6.6	The final optimisation of the locations to determine the global controller location. . .	118
6.7	The location of the controllers in the buildings of CSIR, image from Google Maps. .	125
6.8	The geometric extraction of the controller locations.	125
6.9	The latency of the two re-election criteria, the proposed and the current.	128

CHAPTER 1 INTRODUCTION

1.1 BACKGROUND

The Internet of Things (IoT) is at the centre of the future internet. IoT is an interconnection of many devices, systems, and applications to the internet [9]. According to Cisco, an estimated 50 billion devices will be connected to the internet by the year 2020 [10]. The IoT paradigm is envisaged to permeate industrial manufacturing and production, leading to the Industrial Internet of Things (IIoT) [11]. The IIoT is envisioned to inspire great economic growth and rapid production in various industrial production systems. This digital-oriented industrialisation is termed the “the fourth industrial revolution or 4IR”.

The 4IR is described as the fourth disruptive and major industrialisation trend; an epoch marked by rapid growth and development emanating from automation and data technologies in various disciplines. Some of the major technologies behind this trend include but are not limited to IoT, IIoT, Artificial Intelligence, Virtual Realities, Cyber-Physical systems, Cloud computing, and Cognitive computing. Most of the devices and elements which will participate in the IIoT will be equipped with sensors and actuators; some wireless and some wired. The networking of these sensor nodes extends the scope and purpose of wireless sensor networks (WSNs), whose involvement has always been confined mainly to monitoring. These sensor nodes are small, inexpensive, and intelligent due to the advancement of Micro Electrical Mechanical Systems (MEMS). The major challenges facing industrial systems include the management of the various systems with different proprietary protocols, as well as their sensitivity to time delay, failure, and security.

Software-defined networking (SDN) is a new emerging networking and computing paradigm earmarked as a potential resolution for most of the above challenges. SDN advocates a common standardised

protocol to avoid the challenge of vendor locking [12]. The SDN model separates control and data forwarding in networking elements; and removes the control logic from the network devices and centralises it on a controller [12–16]. The adoption of SDN has gained traction in both industry and academia. Most of the 4IR systems are already applying SDN, including IIoT and WSN as in [17–19].

Software-defined wireless sensor network (SDWSN) is an emerging model formed by applying the SDN model to WSNs. The emergence of SDWSN as a pivot, in stead of WSNs, for the highly anticipated and imminent IoT and IIoT paradigms has ignited much interest and research focus in this area. WSNs are envisaged to play a vital role in IoT as major building blocks [20]. However, WSNs have always been riddled with challenges emanating from their inherent susceptibility to resource constraints which hinder their progress, efficiency, and applicability [21, 22].

The application of SDN in WSNs is also receiving much attention, especially because of its imminent role in IoT [23]. SDWSN is regarded as a potential solution to overcome some of the challenges besetting WSNs while meeting the demands of IoT. With SDWSN, the sensor nodes would be sheer devices with only forwarding capabilities, whereas the control intelligence will be centralised.

1.2 PROBLEM STATEMENT

The application of the SDN model in WSN is set to cultivate the potential of WSNs in modern communication. The SDWSN model will bring about the efficiency that the WSNs have not yet achieved due to their inherent constraints such as limited energy, limited processing power, lack of memory, and limited data rate capacity. The SDWSN model is also envisaged to play a major role in the Internet of Things paradigm which is currently developing at a rapid pace.

1.2.1 Context of the problem

The controller is very central in SDWSN as it holds the intelligence of the entire network. Some of its fundamental functionalities are flow rule generation, mapping functions, and programming interfaces. The flexibility of the SDN model enables dynamic addition of functionalities. The centralisation of the control logic introduces potential drawbacks, detrimental to the efficiency of the network. A central

controller connotes a single point of failure and a potential target for adversaries. The controller could also be overwhelmed by rule setup and other requests from the sensor nodes, resulting in an excessive overhead. Other drawbacks of a centralised controller are issues with access; the distance between the network devices and the controller can negatively affect the performance of the network if not managed properly. Therefore a centralised controller would not be viable in a wireless sensor network, more so considering the inherent challenges such as unreliable links and low bandwidth. Performance and efficiency will also suffer as the network grows.

Most of the research work in SDWSN to date has employed a centralised controller. As the network scales up due to demand, the SDWSN model will have to be scalable, reliable, and efficient, and for that a distributed-control system is a necessity. However, there is work on distributed controllers, albeit for traditional SDN particularly targeted at enterprise networks.

A distribution model based on fragmentation is proposed as a potential solution for the SDWSN control system. This model comprises two-level control architecture consisting of local controllers and a global controller. The challenge is to place the controllers in the network such that there is enough controller coverage for the whole network but also not excessive for costs containment. Thus, an efficient and latency savvy controller placement mechanism for SDWSN is required. Furthermore, as due diligence is accorded in placing the controllers; the same has to be done for controller replacement in an event of controller failure. Most of the current controller replacement methods do not consider location-awareness, thus distance is not considered. In traditional networks, this is not a problem given their abundance of resources, however, SDWSNs are resource constraint and therefore this factor is critical.

1.2.2 Research gap

A distributed-control system for the SDWSN model could alleviate the highlighted challenges, as well as some of the WSN's inherent challenges. A distributed-control system will also come in handy when dealing with the inevitable heterogeneity which models such as IoT entail. This study addresses the deficiencies of a central controller with a distributed-control system which will enable the network to scale, offer redundancy in the event of failure, and improve the overall performance. There are different designs of distributed SDN controllers. However they are not applicable to SDWSN. The

limited resources of the SDWSNs coupled with the kind of data that they carry makes it extremely difficult to design a suitable distributed controller for them. SDWSN requires a controller that is quick, responsive, and process efficient amongst others because it carries data that is sensitive and volatile; lest the data lose relevancy which may lead to unintended, uninformed and catastrophic decisions in active networks. Thus there is a need for an efficient distributed control system for SDWSN which does not compromise any quality imperative.

This study proposes fragmentation as a method of distribution. Fragmentation requires an efficient data consistency model which manages the convergence of data or synchronisation on the distributed controllers. Several data consistency models were studied and eventual consistency was chosen to be properly suitable for the concept of fragmentation. However, for eventual consistency to realise fragmentation, two algorithms were designed and developed. These two algorithms satisfy the principle of the eventual consistency data model through which fragmentation is possible.

1.3 RESEARCH OBJECTIVES AND QUESTIONS

The objectives of this research study are:

- To investigate the feasibility of distributed controllers for SDWSN.
- To propose fragmentation as a method of distribution to achieve efficient SDWSN control.
- To propose alternative algorithms for best-effort and anti-entropy algorithms to achieve a suitable consistency data model for fragmentation.
- To show through evaluation that fragmentation does bring efficiency to SDWSN control.

The research questions that this study aims to answer are:

1. Is a distributed control system ideal for software-defined wireless sensor networks?
2. How can we implement an efficient distributed-control system for software-defined wireless sensor networks without compromising network efficacy?
3. Can we use fragmentation to distribute the control logic of SDWSN?

1.4 HYPOTHESIS AND APPROACH

A distributed-control system in SDWSN seeks to address the challenges of a centralised controller to achieve reliability, scalability, and efficient performance. There are different ways and forms of distribution which are mainly determined by the nature of the network or the data concerned. This study presents an efficient distribution technique suitable for SDWSN control systems. It takes into account all considerations pertaining to SDWSN challenges such as the inherent ills of low bandwidth, energy, and processing, as well as the amount of data exchange or update expected especially from the IoT perspective. The proposed method uses the concept of fragmentation, whereby each cluster segment of the network has its own controller which is lean and very close to the infrastructure elements. There is also a global controller which has a view of the whole network. This two-level control architecture allows a faster response between the sensor nodes and the controller.

Central to distributed systems are the consistency models, which determine the data convergence on the distributed participants (nodes). There are two major consistency models in eventual consistency and strong consistency. The former uses gossip protocols such as anti-entropy and rumour-mongering whilst the latter uses consensus algorithms such as RAFT [24] and/or Paxos [25, 26]. The choice of the consistency model depends on the need of the application and the type of data. This study investigated the applicability of these consistency models and their algorithms in the distributed-control systems for software-defined wireless sensor networks (SDWSNs). An eventual consistency data model was chosen as the most suitable for the proposed system. An eventual consistency data model prioritises availability over convergence. This suits the SDWSN paradigm and will ensure efficient responsiveness to deal with the envisaged rapid data.

The two hypotheses that this study seeks to prove are:

- A distributed-control system is ideal for software-defined wireless sensor networks.
- A fragmentation model brings efficiency to an SDWSN control system.

1.5 RESEARCH GOALS

The goals that this research study seeks to achieve are to:

1. apply the software-defined networking model to improve the efficiency and applicability of wireless sensor networks; and
2. develop a distributed controller to achieve scalability, reliability, and performance for software-defined wireless sensor networks.

1.6 RESEARCH CONTRIBUTION

- The SDWSN model is still in its infancy but it is developing at a rapid pace; therefore, this research undertook a comprehensive and thorough review study of the SDWSN which also focused on the challenges and design requirements. This review study focused on a variety of issues pertaining to SDWSN such as architectures, routing, network management, security, and standards.
- Since the SDWSN model is new and still developing, most of the current research work occurs on the architectural framework. However, the architectures differ mainly because they are mostly narrow based, so that each strives for a particular purpose. Therefore, we designed a new architecture for SDWSN which could be used as a reference for future development of SDWSN solutions. This architecture took into consideration the existing architectures, the current challenges facing the SDWSN model, and the fundamental design requirements which should at least be satisfied for an efficient SDWSN. This architecture is holistic and captures the SDWSN accordingly. It is by far the most comprehensive architecture which will not only serve as a guide for future research in this area but will also contribute to the development of the IoT framework.
- The study then undertook novel research to find a suitable and efficient distribution method for the SDWSN control framework to achieve scalability, reliability, and performance. Different methods, models, and algorithms were studied. A new model, referred to as fragmentation, was identified, designed, and developed.
- The fragmentation model entails a two-level architecture consisting of the local controllers and the global controller. To ensure efficiency in performance and costs, we propose and implement a controller placement mechanism suitable for the SDWSN network.
- The fragmentation model is distributed in its form for redundancy and resilience purposes. To ensure that this redundancy does not undermine the efforts of latency reduction through the optimal controller placement, we propose a controller replacement mechanism which is in line

with the fragmentation ideals. This ensures that during a controller failure, the system chooses the next closest controller as a replacement.

1.7 RESEARCH OUTPUTS

The outputs of this research study are as follows:

1. 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, no. 2, pp. 1872–1899, 2017.
2. H. I. Kobo, G. P. Hancke, and A. M. Abu-Mahfouz, "Towards A Distributed Control System For Software Defined Wireless Sensor Networks," in *Proceedings of the 43rd IEEE conference of Industrial Electronic Society - IECON 2017*, 2017, pp. 6125-6130.
3. H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "Fragmentation-based Distributed Control System for Software Defined Wireless Sensor Networks," *IEEE Industrial Informatics*, vol. In Press, 2018.
4. H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "Efficient Controller Placement and Re-election Mechanism in Distributed Control System for Software Defined Wireless Sensor Networks" *Transactions on Emerging Telecommunications Technologies*, Submitted for review.

1.8 DELINEATION AND LIMITATIONS

The evaluation of this work is based on simulation experimentation, the reason being that the SDWSN paradigm is new and there is a lesser amount of or no ready-made equipment such as SDN-enabled sensor nodes to use for real world experimental testbeds; and there are also cost constraints in procuring such equipment. The study assumes that SDWSN simulation in Cooja is working accordingly.

1.9 OVERVIEW OF STUDY

The rest of the thesis is organised as follows: Chapter 2 presents a comprehensive literature review of SDWSN in general. This chapter delves into all aspects of SDWSN such as the architectural frame-

works, challenges, design requirements etc. In Chapter 3, we highlight the different implementations of controller designs, not only of the SDWSN, but also of the SDN. In Chapter 4, we propose the fragmentation model as a distribution method for the SDWSN control system. This chapter describes the design, all algorithms used and the way they ensure fragmentation. The proposed model is evaluated in Chapter 5. The experimental methodology is described and discussed in detail while the results are presented and discussed. In Chapter 6, we optimise the fragmentation model with a controller placement and controller re-election criteria. Chapter 7 concludes the thesis and presents some of the future research work for consideration.

CHAPTER 2 LITERATURE STUDY

The Software-defined wireless sensor network framework is an emerging concept that resulted from applying software-defined networking to wireless sensor networks. It is still in its infancy and therefore most of the literature is in developmental stages. This chapter discusses the SDWSN literature holistically. It discusses in detail the two models making up the SDWSN; namely the WSN and the SDN. The purpose of this wide consideration is to unravel the current literature, the challenges besetting the SDWSN, the design requirements and the research gaps to contribute to the research direction. The work has been published in a journal article titled “A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements” [1]¹. This had and continues to have a major impact on the research community. This is the first comprehensive survey paper for SDWSN and its contribution is major.

The rest of the chapter is organised as follows. The chapter starts by providing an overview of the SDN model in Section 2.1. This is followed by current challenges in WSNs in Section 2.2, and the potential importance of SDN in addressing WSN’s inherent challenges in Section 2.3. In Section 2.4, we discuss other related networking concepts. Section 2.5 presents a review of the five major aspects of SDWSN: architecture, routing, network management, security and standardisation. Future research challenges and major design requirements for SDWSN are highlighted in sections 2.6 and 2.7 respectively.

2.1 OVERVIEW OF SOFTWARE-DEFINED NETWORKS

Software-defined networking (SDN) is a new networking paradigm that aims to simplify network management and configuration. SDN offers a complete paradigm shift from traditional networking. It

¹ ©2017 IEEE. Reprinted, with permission, from Kobo H.I., Abu-Mafhouz A.M., Hancke G.P., A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements, IEEE Access, February 2017.

seeks to improve network efficiency greatly through high-level novel abstractions. SDN decouples the network intelligence, the control plane, from the packet-forwarding engine, the data plane. The separation enables provision of centralized network intelligence at the controller, which has a global view of the network [27]. SDN introduces benefits such as vendor independence, heterogeneous network management, reliability and security not possible in traditional networks [12, 14, 16, 28]. While SDN was initially earmarked for large-scale enterprise networks, it has the potential to make an impact on any network or computing system. The rest of this section briefly discusses some of the major aspects of SDN, namely the architecture, protocols, standards, applications, and security.

2.1.1 Architecture

Traditional networks, which typically consist of routers and switches as network devices; become difficult to monitor and upgrade as the network grows, thus stifling growth. Large networks also become heterogeneous due to the use of different proprietary protocols, which fundamentally means they consist of different network islands that only cooperate at lower levels of communication [12, 27, 29]. This makes it difficult to implement any policy changes, upgrades and patches. Traditional networks are also mostly hierarchical, tree based and static, which leads to what most have termed “ossification” [12]. Ossification refers to a phenomenon of conforming to the conventional way of networking where everything is coupled on the network device.

An SDN network typically consists of a centralised control plane and highly dispersed data planes (depending on the deployment). The control plane houses the decision-making intelligence of the network, responsible for control and management [27]. After the process of decoupling, the routers become more like data-forwarding switches, with routing decisions made by the controller within the control plane. The controller enables ad hoc management, easier implementation of new policies, seamless protocol upgrades or changes, global visualization, and avoidance of middle-boxes such as firewalls, load balancers, and intrusion detection systems etc. Recent research shows that the deployment of middle-boxes is growing on par with network routers [30].

The SDN architectural framework consists of three layered components, as shown in Figure 2.1. These components are interconnected by various APIs. The first component is the application plane. The application plane interfaces with the various network applications. The second component is the control

plane, which houses the control software. The last component is the data plane (infrastructure plane). This consists of the network devices. The communication between the control and the application planes is defined by APIs, referred herein as the northbound (NB) interface, while the southbound (SB) interface refers to the communication API between the control and the data plane.

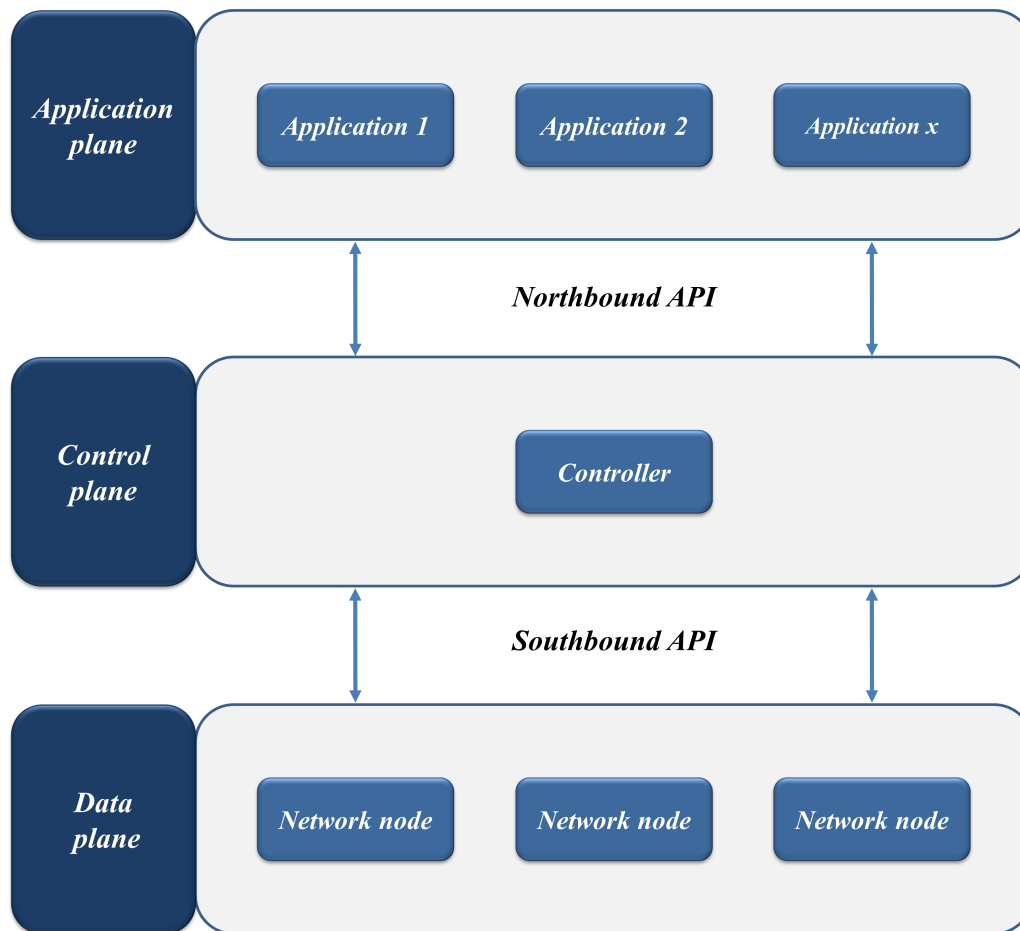


Figure 2.1. The basic SDN framework with the three planes and a central controller [1].

2.1.2 Protocols and Standards

To achieve a high integration of heterogeneous networks, the protocols between the architectural planes need to be standardised. Recent research has focused on the development of both the southbound and northbound API standardisation, with more focus on the southbound API [12]. The standard development organisations are detailed in [13]. The two most popular southbound interface specifications are Forwarding and Control element separation (ForCes) [31, 32] and OpenFlow [3, 33, 34]. They both conform to the principle of decoupling the control and the data plane but are fundamentally different [12].

Table 2.1 below compares the two protocols. ForCes, which is developed by the Internet Engineering Task Force (IETF) working group, consists of two components, the Forwarding Element (FE) and the Control Element (CE). The FE handles the packets while the CE exerts control and executes signal functions and sends instructions to the FE for handling the packet. ForCes uses the concept of a Logical Function Block (LFB), which resides inside the FEs. The LFB has a specific function (such as routing) to process the packets [35], and it enables the CE to control the FE [12].

Table 2.1. Common SDN protocols [1]

Protocol	Standard Body	Communication Protocol	Security	Determinants	API Interface	IP
ForCes	IETF	SCTP	TLS	LFB components	Southbound	IPv4
OpenFlow	ONF	TCP	IPsec	Match fields/Actions	Southbound East/Westbound	IPv4 or IPv6

OpenFlow, which has been developed by the Open Network Foundation (ONF), is by far the most common southbound interface. Although ForCes is deemed to be more powerful and dynamic than OpenFlow, the latter's permeation and adoption in the industry has upstaged ForCes on many fronts [36]. Some literature even regards OpenFlow as the principal [37] and de facto [12] protocol for SDN networks. This research focuses more on OpenFlow due to its prevalence and influence on SDN-based developments. Before the standardisation of ForCes, OpenFlow was the only standardised protocol that allowed direct manipulation of the data plane by the controller [35].

The OpenFlow protocol is flow based, therefore each switch maintains a flow table (which can be altered dynamically by the controller), which consists of flow rules (entries) that determine the handling of packets [31, 33, 38]. Flow entries mainly consist of match fields, counters, and instructions/actions. The match field entry is used to match the incoming packets. The match determinants are the packet header, ingress port, and metadata [3]. Counters collect flow statistics such as the number of received packets, number (size) of bytes and the duration of the flow. The instruction field determines the action to be taken upon a packet match [12]. When a packet is received, the header is extracted, upon which the relevant fields are matched against the flow table entries. If they match, appropriate actions are applied and if more than one entry match is found, prioritisation, based on the highest degree of match, applies [3, 12].

If no match is found, then appropriate rules are actioned, i.e. drop the packet, pass it to the next flow table or send it to the controller for new rules to be made [12]. Figure 2.2 depicts the basic flow of a packet in OpenFlow highlighted by [39,40]. OpenFlow consists of three classes of communication: controller-to-switch, asynchronous, and symmetric [12]. Controller-to-switch is used for configuration, programming, and information retrieval and operates from the controller to the switches. Asynchronous communication is initiated by the switch to the controller and is about packet arrivals, changes, errors, etc. Symmetric communication is sent without the initiation (solicitation) off either the controller or the switch, examples of which are echo packets.

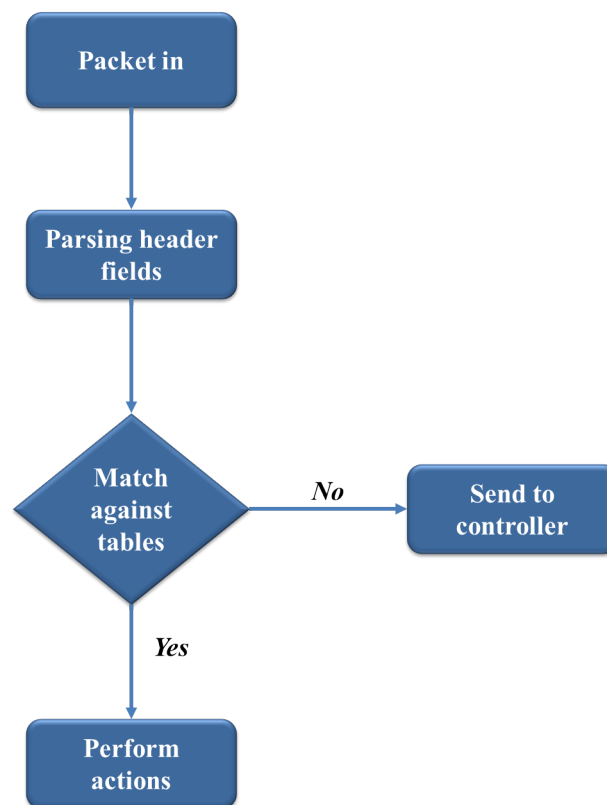


Figure 2.2. Basic packet-forwarding flow in OpenFlow [1, 3].

2.1.3 Applications

SDN has been applied in varied environments such as enterprise campus networks, data centres, and cloud computing services. Enterprise networks are traditionally large and ever expanding rapidly. As the networks grow, the demand for security and better performance increases. Campus networks especially are very dynamic and frequently require a change in policies. SDN simplifies this complex task of making changes by enabling network-wide defined policies to be mapped and programmed

onto underlying network devices. Another advantage of SDN is the elimination of middle boxes, which can now be implemented inside the controller [12] or on the application plane, thus cutting costs of deployment and maintenance [41]. In data centres, many large companies have already implemented SDN architecture to simplify their service provision. SDN has also been used to build private wide-area networks (WAN) connecting data centres successfully, e.g. B4 by Google [42]. The issue of virtualisation and cloud computing services has also necessitated a shift from conventional data storage towards SDN-oriented data storage and management [37].

Other applications of SDN are found in software-defined radios, cellular networks, and wireless networks. In SDN radios, OpenRadio [43] has been proposed, which aims to enable programmability on the PHY and MAC layers [15]. Odin [44] tackles the issue of Authentication, Authorisation, and Accounting in the enterprise Wireless Local Area Network (WLAN) services. SDN is also applied in cognitive radios for dynamic spectrum access [45, 46]. In cellular networks, OpenRoads [7] presents an approach of introducing SDN-based heterogeneity to wireless networks for operators. In wireless networks, SDN has been proposed for use in wireless mesh networks [47, 48]. In campus networks, SDN-oriented OpenFlow networks have been implemented [47, 49, 50]. Another area, which is at the core of this research, is wireless sensor networks [37, 49, 51, 52].

2.1.4 Security

The SDN paradigm presents new security challenges. Centralised control introduces security concerns [53] because networks are susceptible to bugs and other vulnerabilities. Two areas are potential targets [28], namely the centralisation of network intelligence and the software controlling the network. The authors in [54] further identify seven areas of potential security risks in SDNs:

1. Forged or faked traffic flows.
2. Vulnerabilities in switches.
3. Control plane communication.
4. Vulnerabilities of controller.
5. Lack of mechanisms to ensure trust between the controller and the switch.
6. Vulnerabilities of administration station.
7. Lack of trusted resources for forensics and remediation.

Ali *et al.* [29] propose a multi-pronged security response in SDN networks for threat detection, remediation, and correctness to enhance security in SDNs, and as security as a service by which anonymity methods are implemented. Most SDN security measures consist of various verification and validation models. Fresco [55] identifies an application layer and a security kernel on top of the network operating system NOX [56] controller, which interfaces with the security applications and ensures that their policies are implemented. NOX is an OpenFlow controller that allows management and network control applications to be written as centralised programs through a high-level programmatic interface. As multiple applications were implemented, FortNox [57] was developed to alleviate potential conflicts posed by varying applications. It achieves this by setting priorities according to different roles of authorisation. Human administrators are assigned the highest priority, followed by security applications, and then followed by nonsecurity applications [57]. Accordingly, a nonsecurity application would not be able to alter a policy or rule implemented by a human administrator. PermOF [58] is a permission control system used to grant different permission privileges to SDN applications. Changing network conditions affects the network verification measures. Becket *et al.* [59] propose an assertion language based on the VeriFlow [60] algorithm to enable verification and debugging of the SDN applications with dynamic verification conditions. VeriFlow debugs faulty rules inserted by SDN applications before they harm the network. Flover [54] is another verification system that ensures that new security policies do not violate the existing security policies within the OpenFlow network. FlowChecker [61] is a configuration verification model which ensures that OpenFlow rules are consistent within flow tables as well as with other flow rules residing in other switches in a federated network. The SDN security verification solutions are explained in detail in Ali *et al.* [29] and Ahmad *et al.* [53].

2.2 WSN CHALLENGES

It is worth noting that WSNs, despite their great potential, are yet to reach their optimal efficiency. This is largely due to the inherent challenges that they exhibit and the ever-growing scope of demand from applications. These challenges have ignited much research interest in the past decade. Some of the main challenges are listed below.

2.2.1 Energy

Energy conservation is central to the development of WSNs. Since sensor nodes run on a limited battery [21]; it is vitally important to use energy wisely and efficiently to prolong the lifespan of the network significantly. In other instances, the energy source could be replenished through solar and other means. However, as WSNs grow, it could become difficult to replenish the power source which could lead to complete disposition of the sensor nodes as envisioned in [62]. This would, however, depend on the area of deployment and this remains an open research and design consideration. The main cause of power consumption in sensor nodes is largely attributed to sensing, communication and data processing. There have been several attempts at dealing with the energy question in WSNs and this section provides an overview of some of the current main research directions.

There are many types of application in WSNs and each application has its own power needs [63]. The most common approach to reducing high sensing-energy consumption is through sporadic sensing [63]. The sensing unit is used only on demand and is put in idle mode when not in use (inactive mode). Radio communication also consumes a huge chunk of power [64], it involves data transmission and reception. Putting the radio communication in a sleep mode when there is no data exchange efficiently saves a considerable amount of energy. This is called “duty-cycling” [63]. The internal computation of data is another area where energy-saving measures have been applied. The authors in [63] propose that communication should be offset by computation, i.e. reducing the communication overhead by computing more. Different solutions have therefore been suggested to deal intelligently with local data-processing to minimise energy usage. The most common solution is data aggregation [10, 21] where data is internally compressed before it is sent out. Another approach is to disregard redundant data from neighbouring sensor nodes [10]. Mobility-driven approaches have also been suggested, where a specific mobile sensor node collects the data from static sensor nodes and sends it to the controller, thereby saving energy on the static nodes. The topology of the network also plays a crucial role in energy. Sparse placement of nodes uses large amount of energy because the communication range between nodes becomes long [21].

2.2.2 Communication

Communication in WSN takes place through a wireless medium guided by different IEEE specifications operating in the unlicensed industrial, scientific, and medical (ISM) frequency bands. IEEE defines the PHY and MAC layers for Low-Rate Wireless Personal Area Networks (LR-WPAN). IEEE 802.15.1 (Bluetooth) and 802.15.4 are the two most viable protocols for WSNs. These protocols should coexist mutually with other wireless protocols operating in the same ISM band, such as IEEE 802.11a/b/g (WLAN) and IEEE 802.15.3 (ultra-wideband: UWB). WLAN and UWB are not ideally suited for resource-constrained wireless sensors. WLAN and UWB are high-bandwidth wireless communication technologies for devices with high processing power and consistent or easily rechargeable power sources.

Bluetooth is a short-range wireless communication technology based on the IEEE 802.15.1 specification [65]. The earlier version of Bluetooth (Classic) had high power consumption and wasn't entirely suited to LR-WPAN devices. Bluetooth Low Energy (Bluetooth Smart) is an ultra-low power consumption protocol enhancement of the Bluetooth technology, which also increases the communication range [66]. Bluetooth uses two topologies: Piconet and Scatternet. Piconet is formed by one or more Bluetooth devices, referred to as slaves, connected to another Bluetooth device serving as a master. A Scatternet is a cluster of several Piconets. The only drawback of Bluetooth regards its scalability.

IEEE 802.15.4 [2] was developed to address the requirements of the LR-WPAN, particularly ad hoc wireless sensor networks. This standard was proposed specifically for networks of low power consumption, low deployment cost, less complexity, and short-range communication, while maintaining a simple protocol stack [21, 62]. The physical layer supports three frequencies, i.e. 2450 MHz, 915 MHz and 868 MHz [62]. The MAC layer defines two types of node that participate in WSN: reduced functional nodes, which only act as a sensor end device, and full functional nodes, which can act as both the network coordinator and network end device. Network coordinators provide synchronisation, communication, and network-joining services, while end user devices are the actual sensor nodes. Table 2.2 below lists the IEEE 802.15.1 and 802.15.4 specifications, which form the basis of many other protocol standards.

The communication range between the WSN nodes is very short, up to 10 to 20 metres [67]. The data rate is also very low, at around 250 kbps. The low data rate could cause congestion problems especially

Table 2.2. IEEE 802.15.4 specification [1, 2].

	IEEE 802.15.4	IEEE 802.15.1 (BLE)
Layers	Physical MAC	Physical
Frequency	868/915 MHz 2.4 GHz	2.4 GHz
Range	10 to 20 m	10 m to 100 m
Data rate	0.25 Mbps	1 Mbps
Addressing	8 bit or 1 6bit	1 6bit
Devices	100+	7+

in large and highly active deployments, which could affect the overall throughput and latency of the network. Although WSNs are traditionally delay-tolerant, this is likely to change with the introduction of IoT whose applications are sensitive to time.

The ZigBee standard is built on top of IEEE 802.15.4 and defines the communication of the higher layer protocols: network, transport, and application. There has been a lot of work on communication with some researchers even proposing cross-layer communication to save energy [21]. Despite all the great efforts, most communication protocols are yet to propel WSNs to optimal efficiency levels. The TCP/IP protocol, in particular, was considered too heavy to handle for sensor nodes [21, 68], which resulted in addressing challenges. The lack of addressing led to an identity problem, especially in large-scale deployment, until the introduction of the new IPv6 addressing for low-rate wireless personal networks (6LoWPAN). Although that could potentially resolve the intra-WSNs communication, inter-communication remains a challenge in view of the heterogeneity of the IoT framework.

WirelessHart is another WSN standard based on IEEE 802.15.4 for process automation and control [21, 69, 70]. ISA 100.11a is also an IEEE 802.15.4-based standard designed for low data rate wireless monitoring and process automation networks [21, 69]. 6LoWPAN standard enables the IEEE 802.15.4-based devices to communicate using IPv6 [21, 71]. Table 2.3 highlights some of the current IEEE 802.15.4-based standards.

Table 2.3. WSN standards based on IEEE 802.15.4 [1]

Protocol	Layer	Security	Standard Body
ZigBee	Network Application Transport	Link Keys	ZigBee Alliance
Wireless Hart	Network Application Transport	Payload Encryption Message Authentication	Hart Communication Foundation
ISA100a	Network Application Transport	Payload Encryption Message Authentication	ISA
6LoWPAN	Network	Access Control List (ACL) Secure mode	IETF

2.2.3 Routing

WSN topologies are unstructured and therefore many traditional routing protocols are unsuitable. Also, the fact that they are not IP-based makes routing a very challenging yet interesting aspect. The routing is based on the network layer as defined by IEEE 802.15.4. WSN-routing protocols should be lightweight owing to the limited resources that these networks exhibit. WSN's routing protocols are classified as greedy forwarding, data-centric, energy-oriented, localised, and flood based. Greedy forwarding forwards packets to neighbours close to the destination [21]. These kinds of protocols are more effective in dense deployments as opposed to sporadic/intermittent deployments [21]. Data-centric protocols are attribute based, i.e. they are based on a particular attribute such as temperature, and they help to remove redundant data [72]. They normally use compression and aggregation to route packets. Flooding is a common technique in wireless networks where a node reactively broadcasts hello/control packets to its neighbours for possible route determination. It is argued that this method suffers from "implosion", "overlap", and "blindness" [72]. Implosion occurs when redundant messages are received from different nodes. Overlap occur when neighbouring sensor nodes, which observe

the same attributes send similar information. Lastly, they are resource blind because their routing is incognisant of the resource constraints. Energy-efficient algorithm protocols will choose neighbours with high energy levels to route the packets [72]. Localisation-based algorithms use GPS or any other localisation models [73] to localise neighbours in the network and base their routing on those.

The transport layer protocol handles issues such as congestion, packet loss and memory capacity. All these factors, if uncontrolled, waste energy to the detriment of the network. The main goal of the transport layer is therefore to minimise congestion and achieve high reliability [21].

2.2.4 Security

WSNs, like other wireless networks, are susceptible to security threats. Some of the security measures and suitable cryptographic algorithms are discussed in Yick *et al.* [21]. Furthermore, the authors in [67, 74, 75] have identified the fundamental security requirements to be met in WSNs: *Data Authentication, Data Confidentiality, Data Integrity, Availability, and Redundancy*. To achieve the above security goals, WSNs should deal with different threats to which they are susceptible [76]. WSN attacks can be classified into three categories: goal-oriented, performer-oriented, and layer-oriented attacks [77, 78].

Goal-oriented attacks comprise passive and active attacks. A passive attacker monitors and listens to the communication channel and gathers sensitive information, but does not interfere in or interrupt the network operation. An active attacker, on the other hand, monitors, listens, and modifies the data, and thus interrupts the functioning of the network. Some of the active attacks are Denial of Service (DoS), Blackhole/Sinkhole, Wormhole, Hello Flood, Sybil, Modification of data, Node Subversion, Node Malfunction, Message Corruption, False Node, Node Replication, Selective Forwarding, Spoofing, and Fabrication [75, 77–79].

Performer-oriented attacks comprise outside and inside attacks. Outside attacks occur when the adversary exhausts the resources of the node by injecting bogus/unnecessary packets causing a DoS. Inside attacks occur when a malicious node acts legitimately and slowly wreaks havoc in the network. *Layer-oriented* attacks target the different layers of the network stack [75, 78]. Physical layer attacks target the radio operation of the node, either through jamming or tampering. On the Data Link layer

(MAC), the attacker deliberately violates the predefined communication protocol. Network layer attacks target the operation of the routing protocol, i.e. attacks such as Sinkhole diverts all traffic towards an already compromised node. On the Transport layer, the adversary floods connection requests to a particular node to consume its resources. Attacks on the application layer include data corruption and malicious code.

Resource constraints are a major obstacle in implementing optimal security measures in WSNs [21, 78, 80, 81]. There has been a lot of research activity into security counter measures in recent times, with 'low-resource' cryptographic measures at the forefront. Symmetric key cryptography solutions are currently the most preferred, due to their lower implementation cost and their time efficiency. However, there are major drawbacks in that keys are difficult to manage in large networks, with each device requiring a key shared with every other device with which it wishes to communicate. Therefore, if a single key is used and one node becomes compromised, the entire network would be at risk. Although there have been several attempts at improving symmetric key cryptography for WSNs, as highlighted in [53], all the attempts come at the cost of processing resources. Other challenges are their resistance to scalability and the difficulty to implement them in software [75, 78].

Asymmetric cryptography addresses some of the symmetric cryptography drawbacks, e.g. key management would be simplified. However they are considered too heavy and computationally too expensive for the resource-constrained sensor nodes [75, 78, 81]. Recently, more research has focused on toning down these algorithms for WSN nodes as in [78]. The symmetric methods require less computation but they are not robust enough and, on the other hand, asymmetric methods offer potential robustness but at the cost of computation. These are subject to active research in quest for optimal solutions and thus remain an open challenge. With the future direction of the WSNs seemingly converging to IoT, Alcaraz *et al.* [82] note that it is important to have a global perspective of security which does not only focus on WSN but on the entire IoT framework.

2.2.5 Configuration

Manual configuration of any network device is challenging and tedious, especially when the network is growing. WSNs need to respond swiftly to any change in the network and therefore they need dynamic configuration management. Christin *et al.* [74] state that the role of sensor nodes could extend

to offer autonomous functions such as self-healing, and self-discovery, therefore a subtler and more energy-cognisant approach is needed.

2.3 IMPORTANCE OF SDN IN WSN

The advancement of WSNs is thwarted by the inherent challenges that they exhibit. Although much work has been done in an attempt to minimize these problems, there is not yet a holistic solution, as each attempt focuses on a particular problem in isolation. Hence, it is very unlikely that these challenges could be eradicated through the same approach of algorithms and optimisations coupled with the ever-changing specifications and demands of interest. The SDN approach to WSNs is envisaged to potentially solve most of the inherent WSN challenges [5, 27, 68]. The most prevalent and critical WSN problems can potentially be addressed by SDN as follows.

2.3.1 Energy

Energy constraint is a challenge in the development of WSNs and it is by far the most important factor for consideration in this area. Almost all research work in WSNs, in one way or another, attempts to address the issue of energy. Costanzo *et al.* [83] state that SDN in WSN should support common energy-conscious measures as currently being explored in traditional WSNs, such as duty cycling, in-network data aggregation, and cross-layer optimization. The SDN paradigm is handy because through decoupling, the forwarding (switch) nodes are relieved from much of the energy-intensive computational functionalities [27, 84]. An energy-efficient sleep scheduling algorithm based on SDN is proposed in [85]. Most of these energy-consuming functions will now reside in the controller, which has enough power resources. This saves a considerable amount of energy and could potentially prolong the network's life span. Heuristically, this is so but the degree of this assertion (prolonging) is yet to be quantified and remains open for future research.

2.3.2 Network Management and Configuration

Network Management in wireless sensor networks is very complicated and tedious. The network management challenges in WSNs are mostly inherent from traditional infrastructure networks, which include, among other things, provisioning, configuration, and maintenance [27]. The SDN approach

simplifies network management considerably through its simplicity and ability to evolve [27, 83]. In WSN, the reconfiguration and maintenance of the sensor nodes tend to be a complicated and tedious process if management is not flexible; this is also exacerbated by the environments in which WSNs are deployed. These challenges are alleviated by removing the control logic from the sensor nodes, leaving them as mere forwarding elements. These forwarding elements would now be controlled and manipulated from the centralised controller, thereby enabling programmability on the physical infrastructure nodes [27]. SDN also enables a dynamic mapping configuration between the sensor nodes and the controllers (if more than one controller are used) as illustrated with TinySDN [86].

2.3.3 Scalability

The scalability of the WSNs is very important, especially since the advent of the IoT, where SDWSNs are sought to play a critical role. The WSNs become cumbersome as the network grows, to the detriment of efficiency. An abstraction-based model of SDN would aid in keeping the topological organisation and efficacy of the network intact and thus consign the scalability oversight of the WSN to the SDN controller. Although the SDN model traditionally relies on a central controller, many efforts and strides have been made over the years to distribute the control plane; ONOS [87], Hyperflow [88], Difane [89], DevoFlow [90], Kandoo [91], Disco [92], Pratyatsha [93] and Elasticon [94] are some of the distributed SDN controllers.

2.3.4 Routing, Mobility and Localisation

Mobility and localisation are critical for better routing in wireless sensor networks. Depending on the nature of deployment, there could be device mobility, which the network should be able to handle. Normally, traditional routing protocols will periodically update the routing table (proactive) or otherwise source a route on request (reactive) in the event of change. This process is energy intensive and is unsuitable for WSN networks. SDN simplifies this by managing the mobility from the central controller i.e. routing decisions and policies are managed at the controller [27]. Localisation algorithms for example [95], can also be implemented at the controller or at the application plane instead of the resource constrained sensor nodes. This will aid with the discovery of the network topology and consequently better decision-making [27].

2.3.5 Interoperability

WSNs have long been said to be application specific, which leads to resource underutilisation [68]. This shortfall can be resolved by using the SDN approach. SDN alleviates the dependency on vendors by allowing infrastructure elements to be controlled from one central point, thus running one protocol on the elements, albeit from different manufacturers.

2.3.6 Communication

Physical communication is largely managed by the device, but aspects such as media access and duty/sleep scheduling could still be systemwide decisions made by the controller. The duty-cycling functionality could be managed efficiently by the controller. SDN also enhances better control of heterogeneous network infrastructures. Thus, the communication between the SDWSN and other networks could adequately be managed centrally.

2.3.7 Security

The WSN security solutions discussed in Section 2.2.4 above are based on the coupled architecture of the sensor nodes. The SDN's decoupling renders them undesirable for the same purpose. However, these security measures could still play a vital role when they are implemented on the control or application plane. Centralisation of security management simplifies the implementation and configuration of security mechanisms. This global perspective also enables proactive monitoring and evaluation, which leads to quick counter-response in the event of attack, i.e. a malicious node. The traditional SDN security challenges and counter-measures are discussed in Section 2.1.4. These measures apply to WSN to a certain extent because WSNs have unique traits, compared to traditional networks.

A fundamentally important feature of SDN in WSN is the fact that the sensor node becomes a dumb element which only understands controller messages or commands. This makes it difficult and improbable to be used as conduit of malice. Another advantage is the fact that sensor nodes are the peripheral devices in the network; unlike traditional networks where the host computers, which are peripheral could also be security targets. The SDN model also enables flexible configuration in moving

away from the cumbersome and error-prone manual process currently in place. As noted in [16,96,97], configuration errors could lead to security vulnerability.

2.3.8 Summary of discussion

SDN possesses an immense potential for improving network computing, and WSNs, without exception, also stand to benefit. This section discussed the importance of SDN in WSN with the focus on pertinent issues, such as energy, network management, security, configuration, mobility, routing, interoperability, and scalability. SDN therefore indeed deals with most of the inherent WSN challenges and could bring efficiency to WSN. However, there are security issues due to the centralisation of the control logic which could lead to a single point of failure, as well as congestion and overheads concerns, as all decision-making is centralised. These concerns are, however, mitigated by the provision of distributed-control platforms.

2.4 RELATED CONCEPTS

The future of Internet computing is currently a subject of many concerted research efforts in both the academia and the industry. At the centre of this subject is the concept of cloud computing (CC), which has been largely accepted as the next generation of computing infrastructure [98]. Cloud computing, in a nutshell, provides data, computational, and application services to end users in the form of data centres, infrastructure as a service (IaaS), platform as a service (PaaS), software as a service (SaaS), and hardware as a service (HaaS).

2.4.1 The application of SDN in the related concepts

The emergence of the IoT paradigm as a vital role player in the advent of the ubiquitous connectivity of various devices onto the cloud, and the Internet added more complexities which necessitated structural reconsideration. A plethora of IoT devices is envisaged to dominate the Internet space in the years to come. Therefore, the research community and the industry are vigorously leading the quest for architectures and models which would support this IoT upsurge, and one such an enabling model is SDN.

There has been a plethora of studies (with industrial intervention) synergising SDN with CC and IoT such as in [99, 100, 100] and [11, 101–103] respectively. The major characteristics of IoT are low latency, long-lived connection, location awareness, geographical distribution, wireless access, mobility, and heterogeneity [104]. In contrast, cloud computing services inherently have challenges of fluctuating latency, lack of mobility and location awareness [104, 105]. This gave rise to the need to move some of the processing closer to the devices to improve network response efficiency. Fog Computing, conceptualised by Cisco in 2012 [10], extends the cloud computing service provisioning to the edge of the network. It is regarded as a “*highly virtualised platform that provides computation, storage, and network services between end devices and traditional cloud servers*” [104–106].

Although Fog computing refers to “edge” in its definition; it should not, as is often the case, be confused with Edge Computing which in essence extends Fog computing further to the end devices. Thus, it devolves some processing, analytics, and decision-making to end devices and allows them to communicate and share information. Fog computing could also be applied in conjunction with SDN; Truong *et al.* [107] applied SDN with Fog computing for a vehicular ad hoc network (VANET). Fog computing gives rise to other similar concepts in mobile cloud computing (MCC) and mobile edge computing (MEC). MCC “*refers to an infrastructure in which both the data storage and the data processing happen outside of the mobile devices. Mobile cloud applications move the computing power and data storage away from the mobile phones and into the cloud*” [98, 105]. Thus, MCC combines the concept of cloud computing and SDN on mobile phones. MEC, on the other hand, refers to the concept of bringing cloud computing capabilities to the edge of a mobile network closer to the mobile devices for better performance [11, 108]. There are glaring similarities between fog computing and MEC, as well as MCC and SDN. An architecture infusing SDN and MCC is discussed in [109].

MCC is also envisaged to play a key role in the next generation of mobile networks. Aissioui *et al.* [58] propose an SDN controller for 5G mobile cloud management systems. The 5G next generation mobile network is envisaged to be very dense (very close to the end devices) to meet the demand which recently been growing exponentially. Bhushan *et al.* [110] discuss two methods of densification expected to be dominant in the composition of 5G networks: densification of space and frequency. The former refers to a process of densely deploying cells to a spatial locale, called an ultra-dense network [111]. More ultra-dense networks are studied in [112], while Soret *et al.* [113] expound the interference coordination of these networks. The latter refers to the use of large portions of the radio spectrum. This method uses the millimetre wave frequency spectrum [114]. Ali-Ahmad *et al.* [115] and

Trivisonno *et al.* [116] propose different SDN-based network architectures for a 5G mobile network, while Rost *et al.* [117] discuss cloud technologies expected to bring flexibility to the 5G network, including network function virtualization (NFV).

It is clear from the above evolution of wireless networks that heterogeneity will play a huge role in the future of computing and the Internet. Also, the different concepts of computing are not mutually exclusive. Therefore, there is a need for inclusively harmonic coordination and corporation to deal effectively with the data from various sources. SDN and NFV have the potential to ease this integration and thereby bring efficiency to the Internet of Everything [118].

An SDN-based vehicular ad hoc network (VANET) with fog computing is explored and implemented in [109]. The architecture used an SDN controller, sitting above the fog, but connected to the cloud. The fog is composed of an SDN road-site-unit controller (RSUC), SDN RSU, SDN wireless nodes (vehicles) and a cellular base station (BS). The communication between the controller and the RSUC and between the RSUC and RSU takes place through a broadband link. The vehicle nodes use long-range cellular networks such as 3G, WiMAX, and 4G LTE to communicate with the BS whilst they use a wireless connection or wave to communicate amongst themselves. Anadiotis *et al.* [119] define an SDN operating system for IoT that integrates SDN-based WSN (SDN-WISE) and an SDN-based Ethernet network using an ONOS controller. This experiment shows how heterogeneity between different kinds of SDN network can be achieved.

The above studies demonstrate that different kinds of network and system can co-exist. Systems such as smart grid [120], smart water system [121], smart cities [122], health care [123], Vanets [107, 124] etc. are expected to grow explosively, with Cisco estimating a total of 50 billion devices by 2020 [10]. Most of these systems will be equipped with sensors (mostly wireless) which collects data from the ground, as Anadiotis *et al.* [119] note that they are “fundamental ingredients to the IoT ecosystem”. It is for this reason that this study focuses on the application of SDN direct from the bottom devices. This will ensure a bottom-up approach and consistent architectural application of the SDN principles.

Figure 2.3 depicts the technologies envisaged to feature predominantly in future computing of IoT in an inverted pyramid format. The technologies above the “Internet of Things” perimeter are considered carriers of the IoT, whilst those below are the building blocks of the IoT network. The SDN model should ideally be applied throughout, i.e. from the bottom devices to the top technologies. Recent

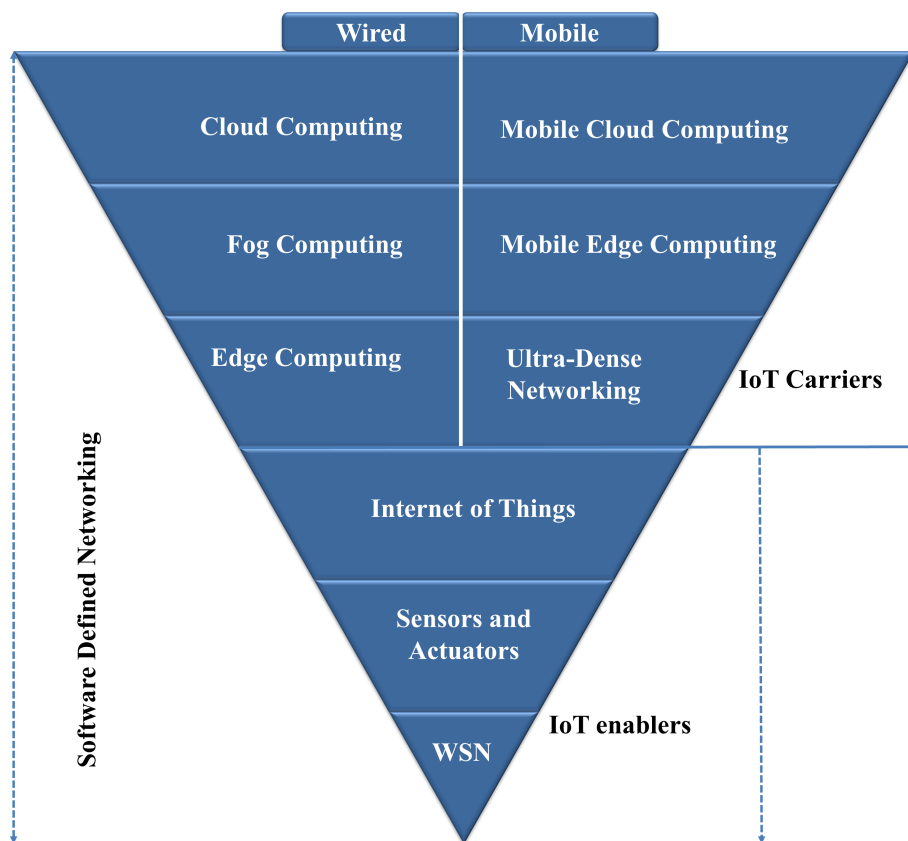


Figure 2.3. Current and future network computing technologies with SDN [1].

studies show that the application of SDN has gained much of traction from the top pyramid technologies. In contrast, the application of SDN on IoT devices is still lagging behind. Although progressive strides have been made; they are still very much in a developmental stage. The survey study below provides a state of measure of the extent of SDN applications: SDN in traditional networks [12–14, 37, 125, 126], SDN in wireless networks [15, 125, 127], SDN in cloud computing [100, 128, 129], SDN in Fog computing [105, 106], SDN in mobile cloud computing [98, 130], SDN in IoT [129, 131]. At this stage, there is no survey focusing on the bottom devices such as sensors. This chapter fills this gap by surveying recent work on the application of SDN in WSN, and the set of design requirements which should be considered for the development of the SDWSN paradigm. Although some of the sensors from the IoT perspective will be wired, a considerable amount of literature shows that WSNs have an extensive role to play. Vaquero *et al.* [132] concur, and further identify mobile devices and sensor/actuators as the major driving force behind the growth of IoT, with the sensing devices estimated to surpass all. Also, WSNs exhibit a special set of challenges that warrants urgent attention.

2.4.2 Summary of discussion

IoT is at the centre of future Internet computing as the connectivity of various distinct elements is intensified. As WSNs are envisaged to play a pivotal role in this IoT upsurge, there are many other related concepts which are intrinsically imperative. These concepts can be summarised categorically as carriers of the IoT traffic: the wireless mobile networks (Wi-Fi, LTE, 5G, etc.); computing frameworks: cloud computing, fog computing, mobile edge computing, mobile cloud computing, as well as the building blocks: sensors and actuators. This section discussed the application of the SDN model to all these concepts, including the IoT. The relationships amongst them were also highlighted and elucidated through a diagram. Also highlighted is the need for a SDWSN review to provide an indication of the state-of-the-art research in this field, which is lagging behind the other concepts. Therefore, a bottom-up approach to the application of SDN is suggested to realise heterogeneous IoT.

2.5 SOFTWARE-DEFINED WIRELESS SENSOR NETWORKS

2.5.1 SDWSN Architectures

The SDN approach to wireless sensor networks entails abstracting different functionalities and reorganising them along the three logical planes of the SDN model: application, control, and data. The development of the SDWSN architecture is still in its infancy, but valuable inroads have been made by the research fraternity. Although there are different implementations of the architectures, they all conform to the fundamentals of SDN: decoupling. Figure 2.4 depicts the basic functionalities of SDWSN as applied by various researchers.

2.5.1.1 Forwarding devices

The sensor nodes are the basis of the infrastructure layer or the data plane. The sensor nodes comprise hardware and software components. The hardware consists of a power unit, sensing unit, and radio. The hardware components are on the physical layer (PHY), which together with the MAC layer performs the IEEE 802.15.4 functionalities as specified for LR-WPAN [2].

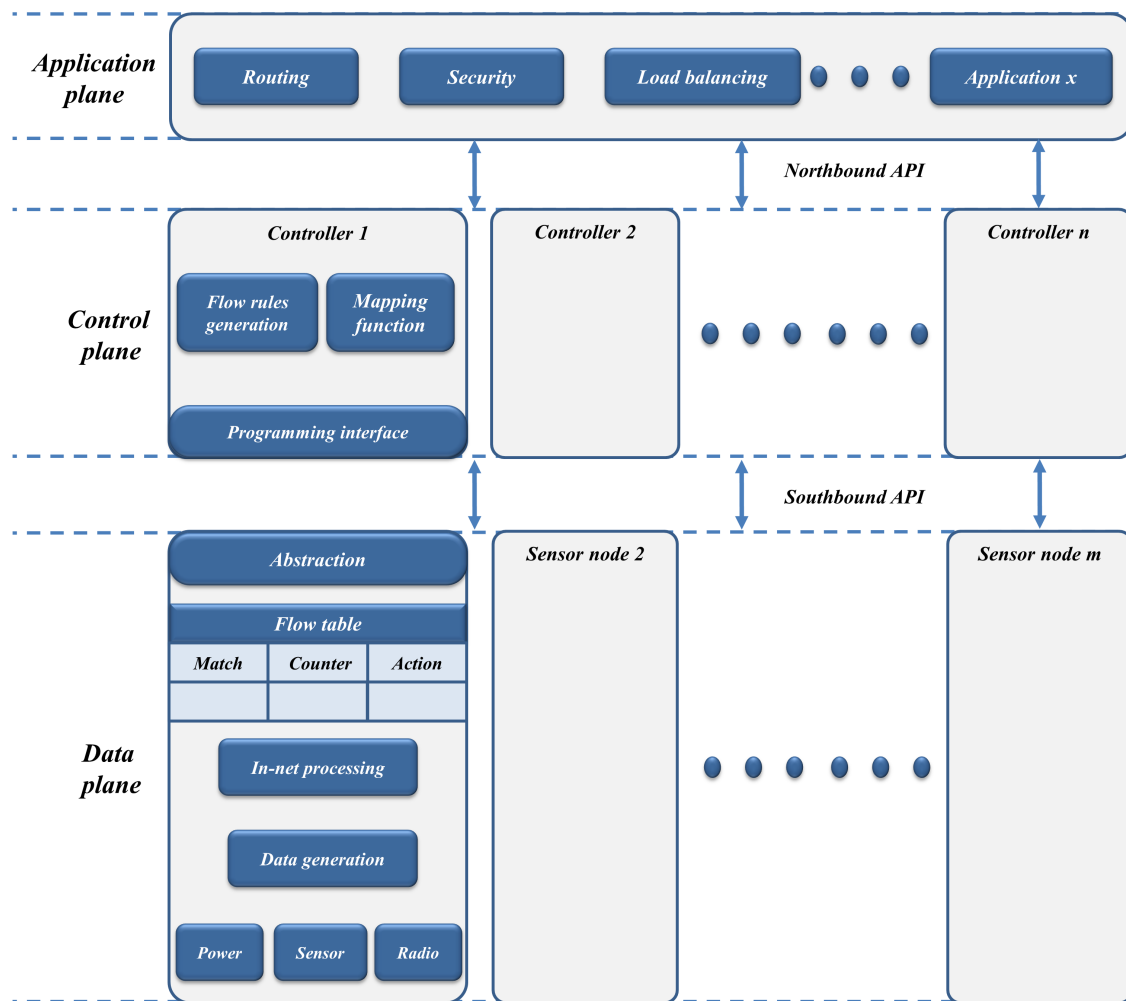


Figure 2.4. Basic SDWSN architecture as currently applied by various studies. Different functionalities are distributed along the three planes [1].

- **MAC and PHY**

The IEEE 802.15.4 functionalities are pertinent to the communication of the SDWSN wireless networking. The architecture proposed by Costanzo *et al.* [83], namely SDWN, consists of a generic node and a sink node. The generic node consists of three layers: PHY, MAC, and Network Operating System (NOS). The sink node, on the other hand, has two parts, the generic node and the controller, which are serially connected. Jacobsson *et al.* [20] propose a reconfigurable PHY layer to enable flexible configuration and altering of parameters. The MAC layer handles node identification, which is inherently a major challenge in WSNs. Luo *et al.* [68] propose sensor OpenFlow, which aims to address the identification (addressing) problem

which emanates from the fact that WSNs are data-centric and attribute based as opposed to the Ethernet-based networks (address-centric) [21, 68]. To alleviate the identification challenge, the authors aim to redefine the addressing model by using a ZigBee 16-bit network address and using concatenated value pairs, which entail using the attribute descriptions e.g. all packets with a temperature of a certain threshold.

Another method proposed is to augment the WSN by IP addresses, which many deem to be infeasible for sensor nodes. However, several studies recently, for example Mudumbe *et al.* [73], and Pediredla *et al.* [133], have considered the deployment of an IP-based WSN using different technologies such as 6LoWPAN (IPv6 over low-power wireless area network) [134] protocol. Despite IPv4 being ruled out on WSNs, IPv6-based 6LoWPAN uses compression for the much constrained IEEE 802.15.4 devices. 6LoWPAN is earmarked as a pragmatic enabler of the Internet of Things [71, 133, 135]. An IP-based WSN may be considered as an alternative solution. Mahmud *et al.* propose Flow-Sensor [51] which aims to achieve reliability by separating the propagation of the control and data packets. The control packets use OpenFlow, while the data packets use the normal TCP/IP.

- **Data-processing**

The software part of the sensor node typically consists of a flow table, sensing element, in-network processing, and an abstraction layer. The sensing module generates traffic which is then passed to the processor. The in-network processing could either be data aggregation or data fusion. Ideally, aggregation would be at the sensor node while fusion takes place at the sink node. The flow tables store rules as prescribed by the controller, while the abstraction layer provides an interface for communicating with the controller. The software component of the sensor node plays a critical role in the processing of the sensed data and routing functionalities. The processing of the sensed data is carried out differently by various research works. Luo *et al.* [68], Constanzo *et al.* [83], and Gallaccio [136] recommend data aggregation for in-network processing. In the cognitive SDWSN architecture proposed by Huang *et al.* [137], information fusion is employed. They further use over-the-air-programming (OTAP) [65] to distribute routing information.

2.5.1.2 Controller

The controller in SDWSN plays a very critical role as it holds the intelligence of the whole network. Its fundamental functionalities are flow rule generation, mapping functions, and programming interfaces. The SDN model allows for more functionality to be added. There are different implementations of the control plane: it could be centralised [56, 138–140] or distributed [87, 88, 92]. The use of centralised controllers in addition to local controllers is considered in [89, 90, 141]. Additionally, elastic solutions [58, 93, 94, 142] dynamically add or reduce controllers according to network load.

The SDWN architecture in [83] uses an embedded controller in a Linux system which is attached to a sink node through a serial connection. The sink node communicates with other sensor nodes on a wireless interface. The embedded system consists of an adaptation module, a virtualiser, and a controller. The adaptation module is used for message formatting. The virtualiser ensures that all information collected is collated to form a virtualisation of the network state to the controller [83, 143]. A major drawback of this approach is the fact that the serial communication between the sink and the controller limits the controller and the sink to a one-on-one relationship. This also poses a problem of scalability. Such architectures will only be viable in a small controllable network.

In other instances, the sink node also acts as a base station (BS), which also houses the controller, as in [144] and [27]. De Gante *et al.* [27] propose a BS based SDWSN which consists of five layers: physical, medium access, NOS, middleware, and application. The middleware layer consists of a controller, mapping function, mapping information, and a flow table's definition. The controller controls and manages the network and keeps the state and topology of the network up to date. It uses monitoring messages to acquire network information, such as sensor node energy levels, distance from the BS, node's neighbour list and link status parameters, such as link quality, response time, etc. The mapping function processes the monitored data from the sensor nodes and creates a network map (topology view). The information acquired is stored in the mapping information module. The purpose of the application layer is to locate the sensed area, and therefore it consists of location and tracking algorithms to maintain accurate information about the node's position.

Olivier *et al.* [144] propose a cluster-based SDWSN architecture, which also has a base station. They apply the SDN model in a clustered WSN for the formation of what they refer to as a software-defined cluster sensor network (SDCSN). The sensor network is organized in clusters (SDN domains)

comprising sensor nodes and a gateway or cluster head. The cluster head herein referred to as SDN cluster head (SDNCH), controls and coordinates all sensor nodes in its domain. The SDNCH can implement its own security policies and thus secure the domain against outside attacks. The SDNCH has a partial view of the network and communicates with other SDNCHs through the gateway. It uses WE-Bridge [145] to exchange local data with other SDNCHs.

Huang *et al.* [137] propose a cognitive SDWSN framework to improve energy efficiency and adaptability of WSNs for environmental monitoring. The architecture employs a reinforcement learning method to perform value redundancy filtering and load-balancing routing to realise energy efficiency and adaptability of WSN to a changing environment [137]. In TinySDN multiple controllers are used. The prototype consists of a sensor node and a sink node attached serially to a controller. The SDN sensor node must first find a controller to join using a Collection Tree Protocol (CTP). The CTP protocol is also used to send information, such as link quality to the controller. The link quality in TinySDN uses link estimator instead of RSSI (received signal strength indicator), even though they acknowledge that RSSI is more accurate, the link estimator is chosen as it is hardware independent. The framework was implemented and tested by using a Cooja simulator [146].

A hybrid control model is employed in [20] where there is a main controller at the control plane and a local controller on each sensor node. The purpose of the local controller is to setup, reconfigure, monitor and execute instructions from the controller. The sensor nodes are equipped with virtual machines (VM) which help in installing new protocols or code patches when needed. Changes can be installed using native code and dynamic linking for homogenous networks and/or byte code with VMs for heterogeneous networks.

2.5.1.3 Tools

TinySDN is proposed in [86] and implemented like SDWN [83]. TinySDN is based on TinyOS [147, 148], a sensor-network-based OS. TinySDN aims to reduce the control traffic. The TinySDN architecture consists of two nodes, an SDN sensor node and an SDN controller. Sensor OpenFlow is proposed in [68], which there is a communication protocol between the control plane and the data plane [149]. Sensor OpenFlow is based on OpenFlow [128], which until recently was earmarked for

enterprise and carrier networks [59]. Sensor OpenFlow also aims to achieve compatibility with other OpenFlow-based sensors.

TinySDN was evaluated using a Cooja simulator. Cooja [146] is a simulator tool used to simulate sensor motes running Contiki OS. Mininet [150] is the most prevalent simulation tool used for SDN OpenFlow networks. However, Estinet [151] is purported to be better than Mininet in terms of performance and scalability though it remains proprietary [85]. Fs-SDN [49] is another SDN simulator which was developed to facilitate SDN controller application prototyping; POX [152] controller was used in development. Son *et al.* [153] developed CloudSimSDN to simulate and test SDN cloud services, since Mininet and other simulators cannot simulate cloud-specific features. Other simulators that support OpenFlow protocol are NS-3 [154] and Trema [155] which are implemented in C++ and C (and Ruby) respectively [156].

2.5.1.4 Internet of Things enabling architectures

The SDWSN architecture proposed by Jacobsson *et al.* extends this novelty to the new Internet of Things (IoT) paradigm in which WSNs are envisaged to play a huge role [20, 102]. There has been more work in trying to integrate SDWSN aspects into the IoT framework. The authors in [51] also extend their work to consider the use in IoT [157]. Perhaps a more complete IoT architecture is given in [103], where the authors apply SDN principles in IoT heterogeneous networks. Hakiri *et al.* [158] outline an IoT architecture that synergises SDN with Data Distribution Services (DDS) and highlights some of the challenges pertaining to standardisation, mobility, network gateway access and QoS support. Hu [11] introduces a new architectural paradigm in the Industrial Internet of Things (IIoT), which focuses on industrial production systems. It calls for a detailed feasibility study on the synergy of SDWSN with other wireless networks in the context of IIoT.

Zeng *et al.* [38] propose an architecture model that combines SDWSN and cloud computing. In a cloud computing service provision, e.g. IaaS, sensing is offered as a service to different applications. A sensing request is sent to the controller, which will subsequently send the request to a suitable WSN(s) which offers that sensing service. Many WSNs exist in physical space and SDN is used to normalise them into one integrated common network of SDWSN. The SDN model is used to enable flexible

alterations to meet the dynamic sensing requests (by different applications). The sensed data can also be combined with other cloud service data for mashup services.

Table 2.4 highlights a compact comparison of the different architectural frameworks stated above. The implementation attribute (column) checks whether the proposed architecture has been implemented (simulation or testbed) or not. The controller attribute checks whether the controller is distributed or centralised. The in-network processing checks whether there is any form of processing on the nodes. Finally, the main purpose of the architecture is listed in the focus attribute.

Table 2.4. Current SDWSN architectures [1].

Architecture	Implementation	Controller	In-Processing	Focus
Sensor-OpenFlow [68]	Simulation Testbed	Central	✓	Protocol (SB API) Identification
SDWN [83]	Proposed	Central	✓	Generic
Flow-Sensor [51]	Simulation	Central		Reliability
SDWSN [27]	Proposed	Central		Management Mobility Localisation
TinySDN [86]	Simulation	Distributed		Multiple controllers
SDWSN [20]	Proposed	Central	✓	Architecture/IOT In-network processing
Cognitive SDWSN [159]	Simulation	Central	✓	Energy Efficiency Adaptability
SDCSN [144]	Proposed	Distributed	✓	Controller placement
SDSN [38]	Proposed	Distributed		Sensing as a service
SDN-WISE [136]	Testbed Simulation	Distributed	✓	Reducing overheads Finite state SDN-WISE

2.5.2 Routing

It is worth noting that in SDN's paradigm, the routing functionalities are logically centralised at the controller. Traditional WSN routing protocols run on the nodes and consume large amount of energy.

Such approaches in their current state are not viable for SDWSN; however, their algorithmic concepts could be implemented at the controller level, thus requiring a transformation from an infrastructure-based to a software-based approach. The traditional WSN routing methods, as discussed and classified in [5] remain to be tailored for SDWSN based on SDN guidelines. The most related current work to date is found in the routing model architecture proposed by Shanmugapriya and Shivakumar [4]. The authors combine context-aware and policy-based routing modules in line with the SDN principles. Figure 2.5 depicts the proposed architecture.

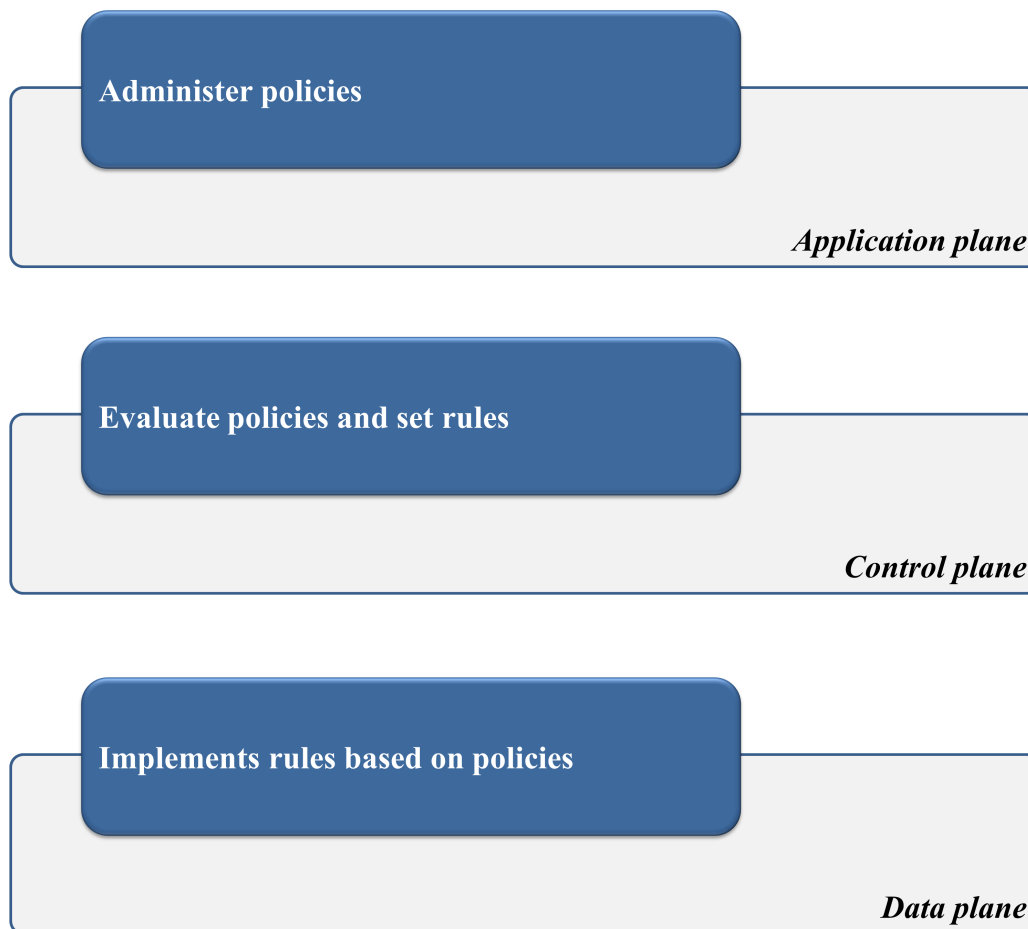


Figure 2.5. The context-aware and policy based routing model [1, 4].

The model has six phases:

1. *Initiation phase:* Sensor nodes connect to the controller using the Sensor OpenFlow protocol. The nodes also supply the controller with all relevant information such as node status, CPU load, etc.

2. *Administration Phase*: This phase entails the defining of rules which will ultimately be used for routing decisions.
3. *Topology discovery phase*: The node supplies the controller not only with its own information but its neighbours' as well. The controller forms a routing map table. The table has all the nodes and their next best hops. The route link has context information such as CPU load, service information, and power levels of the next hop. Services are any policies defined for that route such as security, privacy, etc. [4].
4. *Decision phase*: The controller uses a recursive destination based lookup algorithm to look for a route in the mapping table.
5. *Policy-based route phase*: The routes are chosen by using defined policies. The algorithm lookup will determine all available routes towards the destination and a particular route will be chosen if it matches the policy criteria, i.e. ignore any route with a CPU usage of at least 90%. Should all routes not match the policy, packets could be dropped.
6. *Enforcement phase*: The controller enforces the routing onto the switch devices.

Han *et al.* [5] propose a cluster-based routing protocol based on SDN. They set an OpenFlow-oriented SDN network based on three types of nodes: master node, centre node, and a normal node where the master node is a controller, the centre node is equivalent to a switch/sensor node (packetforwarding device), and the normal node is only for receiving data. The SDN network uses a NOX controller. They further highlight the use of FlowVisor [160, 161] for controller scalability purposes. FlowVisor is an SDN based-virtualisation application model which enables multiple controllers to use or manage one switch concurrently. Each controller accesses the switch through a dedicated virtual portion called slice. Like all the OpenFlow-based routing, the master node determines the route upon receiving a new request. The routing or the forwarding policy is based on the QoS information. The centre nodes maintain the flow tables. The network is arranged in clusters, with the centre nodes as cluster heads which then communicate with the master node via a secure channel, as shown in Figure 2.6. The location of the centre node is critical and therefore it uses a distance-aware routing algorithm based on content addressable network (CAN). The authors highlight limited energy on the nodes as a challenge.

Yuan *et al.* [6], propose a hybrid routing model which combines the ideals of an ad hoc on-demand distance vector (AODV) [162] protocol with OpenFlow. AODV is a wireless and mobile ad hoc

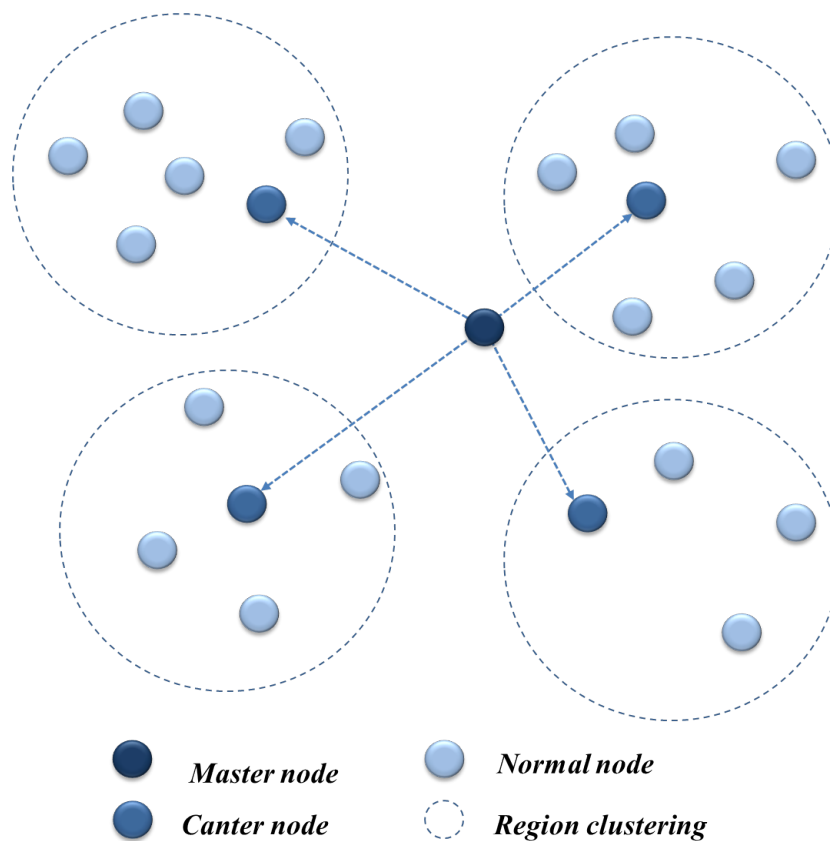


Figure 2.6. SDN cluster-based routing [1, 5].

network routing protocol. They use an AODV daemon to implement the AODV algorithm as well as OpenvSwitch [163] as an OpenFlow agent which enables the communication between the devices and the controller [6]. Figure 2.7 depicts the system architecture, which was implemented on a physical device (Raspberry Pi).

Most of the current efforts lack qualitative and quantitative evidence of their efficiency in a real network environment. Their performance should be tested against common network factors such as latency, QoS, packet delivery, congestion etc. The current SDWSN-routing protocols are listed in Table 2.5.

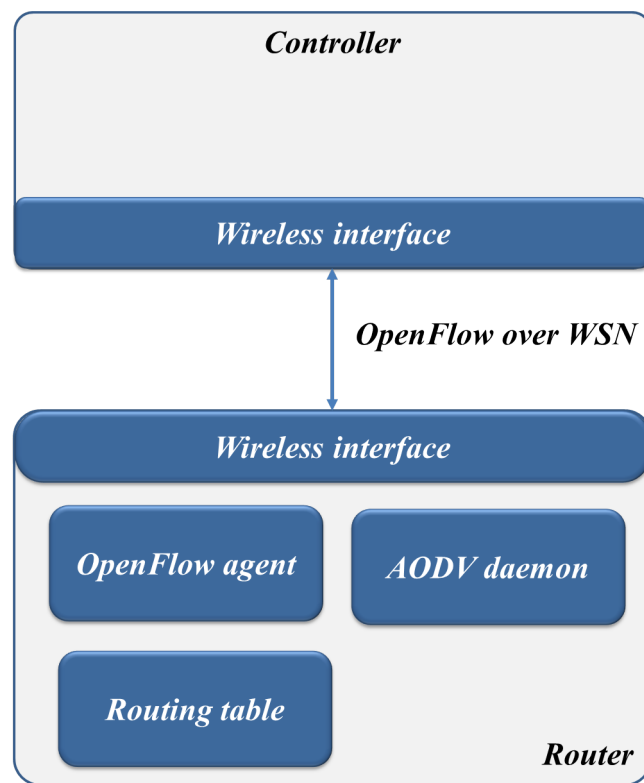


Figure 2.7. The hybrid routing model architecture [1, 6].

Table 2.5. Current SDWSN-routing protocols [1]

Name	Algorithm	SDN layer	Testing
Shanmugapriya <i>et al.</i> [4]	Context aware and policy based	Application	proposed
Han <i>et al.</i> [5]	Cluster based (CAN)	Controller	Matlab
Yuan <i>et al.</i> [6]	Link State (AODV)	Controller	tshark

2.5.3 Network Management

Network management creates a platform for service management applications to run and configure various services with ease, i.e. new policies, patches, and new code. It also defines the interface between the controller and the applications, therefore northbound in accordance with the SDN model. As Kreutz *et al.* [28] and Vaquero *et al.* [132] allude, there has been less focus on the NB interface, whereas much attention has been devoted to SB interfaces. The northbound interfaces are mostly built on top of the network SDN controllers. There are several SDN controllers which support OpenFlow

as highlighted in Chapter 3 to follow. These controllers were built for traditional SDN networks. Though some of these controllers have been successfully used in wireless networks, they remain to be implemented and tested for SDWSN. While the controllers provide low-level control over the underlying network devices, they are supported by network programming languages which translate high level network policies into forwarding rules. Procera [164, 165] is a declarative language which also couples as a network control framework. More SDN-based network declarative languages can be found in [166–169].

OpenRoads [7, 50] is the most common northbound API that allows network applications to define and implement any policy framework on the switch through the controller. OpenRoads, also called wireless OpenFlow, is a network management interface that allows management of various wireless networks using virtualization [7]. OpenRoads uses FlowVisor to slice the OpenFlow network and provides a proper isolation amongst the slices, thus enabling multiple controllers as envisaged in [160]. Each controller is responsible for its slice. This also enables multiple experiments to run concurrently on the network [7, 50]. OpenRoads uses OpenFlow to control and manipulate the OpenFlow-based switches and uses SNMPVisor (based on SNMP protocol) to configure the devices [7]. However, the newer releases of OpenFlow come with OF-CONFIG, which configures and manage the switches [3]. Figure 2.8 shows the architectural stack of the OpenRoads deployment.

The OpenRoads [50] platform was used in a heterogeneous wireless network consisting of Wi-Fi APs, Wi-Max base station, and Ethernet switches; all of which run the OpenFlow protocol. The network was used to test various mobility management applications. The OpenRoads architecture is built on top of the NOX OpenFlow controller [7, 50] which controls the Wi-Fi APS, Wi-Max base station and Ethernet switches, as well as the SNMP protocol, to have control over the power, frequency, data rate, SSID, etc. of the device elements [7].

Huang *et al.* [137] propose an SDN-based management framework for the IoT devices. As the 6LoWPAN protocol gains momentum, particularly in view of the IoT, new network management solutions are developed to cater for these devices. Feng *et al.* [170], LNMP [171] and 6LoWPAN-SNMP [172] are Simple Network Management Protocol (SNMP)-based network management solutions for 6LoWPANs. SNMP is a well-known and commonly used network management protocol in enterprise networks and its ripple effect on the development of new management architectures comes as no surprise. Although this model was used in different wireless networks, it provides the SDWSN

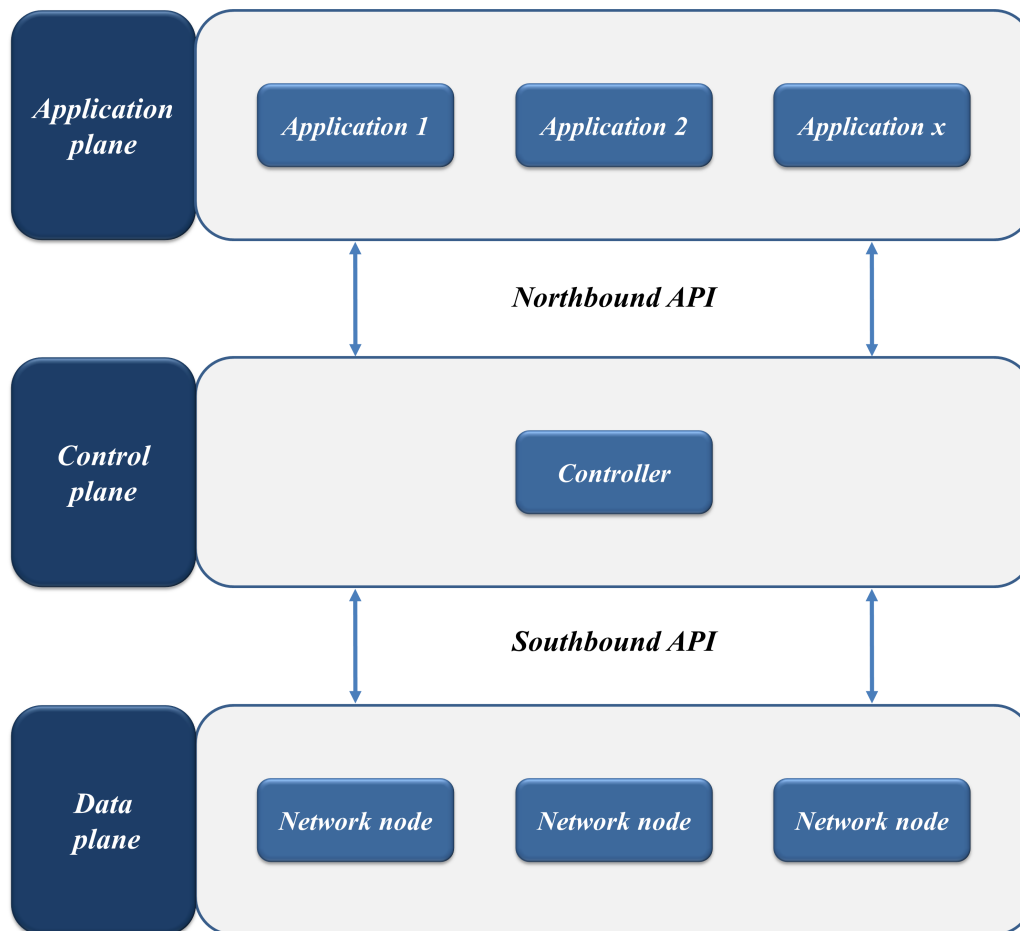


Figure 2.8. The OpenRoads architecture [1, 7].

with a proper model of network management, especially considering the potential of heterogeneity in SDWSN, and thus remains open in respect of SDWSN. The challenge for northbound interfaces is the fact that they are not standardized and therefore a plethora of different incompatible designs is probable [3].

2.5.4 Summary of discussion

The SDN model in WSN has been embraced with vigour and enthusiasm. This section discussed the current state-of-the-art research into SDWSN with special focus on architecture design, network management and routing. There has been huge progress in advancing the SDWSN model, although it is still in its infancy. The architectures studied above vary slightly in implementation, affirming that consensus on the design has yet to be reached. On the forwarding nodes, the functions of the

MAC and PHY layers remain in WSNs. However, there are disparities in the way processing of data is implemented across various research studies. The implementation of the control logic also varies; some have it at the sink while others have it at a level higher. Ideally, the end-to-end network management would include the two interfaces, southbound and northbound; but the NB appears to be lagging behind as far as SDWSN is concerned. The OpenFlow protocol is most prevalent between the sensor nodes and the controller. On the other hand, there is no commonality with the NB interface as different control platforms use their own interfaces for this purpose. However, REST APIs are envisaged to play a pivotal role, although not thoroughly tried in SDWSN.

2.6 FUTURE RESEARCH CHALLENGES

Most of the existing SDWSN architectures differ in certain respects. However, they share fundamental commonalities of which most are influenced by the OpenFlow protocol. While there seems to be converging consensus about the application of OpenFlow, it is yet to be adequately proven for SDWSN. This section looks at some of the shortcomings of the reviewed SDWSN architectures, while aligning them for future research considerations.

2.6.1 WSN-inherent challenges

Most WSN-inherent challenges are also not yet adequately addressed. Although the SDN paradigm promises a huge reduction in energy consumption by the nodes, the extent of this needs to be evaluated and quantified. The local in-node processing and its impact need to be scrutinised in detail and . The amount of processing needed in proportion to energy usage should be determined. Processing is very critical in alleviating issues such as implosion, redundancy, etc. As far as processing is concerned, when coupled with energy considerations, what amount of processing should be left on the node? The trade-off between internal processing and controller processing needs to be evaluated to address the issue of aggregation too. The aggregation of the sensed data is also paramount and needs closer research attention. Another problem is processing of the different data attributes that these sensor nodes represent. The aggregation problem in heterogeneous networks also needs to be explored to include the model of abstraction.

Transmission of the data is also a concern. It might not be practicable to have all sensor nodes transmitting their raw data to the controller as that will result in excessive delay and congestion in the network. As some of the above architectural designs have in-network processing on the sensor nodes; this has to be tested for its efficiency. Having a local controller on the sensor node has been suggested in [20] while others suggest having a sink node which houses the local controller.

Most of the switches used for SDN purposes use ternary content-addressable memory (TCAM). Memory is one of the scarce resources in WSNs and it remains to be seen whether TCAM would work or even be affordable. The memory management techniques explored in [173] could be investigated from the SDWSN perspective. The other question that begs research attention is how the sensor nodes dynamically join the network, and also what happens when a change is made when a particular node is down.

2.6.2 Network operating system

There seems to be disparity about the use of the network operating system and a functional protocol. Some implementations above seem to be more inclined to an OS which includes some basic functionalities of a protocol and likewise the protocol-based architectures seem to include some functionalities of an OS. This stand-off needs proper evaluation, to determine if the two should co-exist or operate independently.

2.6.3 Practical implementation and evaluation

Although there have been research efforts that made inroads in tailoring OpenFlow for SDWSN, there has been no practical application, because most are simulated or are just a proposed general framework. The practicality of the SDWSN would provide a clear indication of the progress made to date. That would also present an opportunity to evaluate common wireless network issues such as QoS, reliability, packet loss, bandwidth, stability, efficiency, scalability, etc. Although Luo *et al.* [68] and De Oliveira *et al.* [86] propose an architecture for the physical sensor nodes, they were only tested through simulation and therefore there is a need for an actual prototype of the SDN-based sensor node.

2.6.4 Inter and intraplane communication

The communication between the controller and the application layer is important for the overall structural security of the network. Hence, any protocol considered needs to address the security impasse adequately. The communication between the controller and the infrastructure devices, the southbound interface, is also crucial because it is the enabler of the transition from the resourced control plane to the less resourced data plane. This transition presents an open research problem to be explored. The communication amongst the sensor nodes also needs to be defined properly when TCP/IP (IPv4) is ruled out [51], therefore, the exploration of 6LoWPAN needs to be intensified.

2.6.5 Standardisation

The architectures also differ fundamentally in the allotment of different functionalities along the two SDN control and application planes. Most of these architectures focus on one or two entities, hence there is a need for a holistic architecture which covers all building blocks of the model. The lack of standards in SDWSN could derail the development and further exacerbate the issue of dependant compatibility, which the SDN model seeks to avoid.

There is an urgent need for SDWSN standardisation. The lack of it would result in different incoherent and incompatible architectures, which goes against the SDN's principle of heterogeneity. Whilst a formal standard for SDWSN is yet to emerge, the standardisation of its constituents, WSN and SDN, has been developing at a pace, IEEE 802.15.4 [67], ZigBee [2], and ONF [35], IETF [31] respectively. It is not yet known whether conformation to these two standard groups would satisfy the requirements of SDWSN, or perhaps a new standard would be necessary. The standardisation of the IoT framework is also imperative and has seen standardisation of its constituent networks [174]. Hence there is a need for a holistic standard or specification, which will guide the compliance of future networks.

2.6.6 Security

Security is a very critical architectural consideration, especially in this cyber age, where everything is envisaged to be connected. However, security in SDWSN is still a very open area that is yet to receive much attention. Most of the work in SDWSN is still very much on the architectural framework,

partly due to the infancy of this field. But nonetheless, since SDWSN synergises two areas, we should draw reference from their respective work. The security structure of the model needs to take care of the SDWSN network itself, the controller, the sensor node device, and the communication protocols among other things.

Most research work addresses security in traditional SDN and WSN networks. Some of these concepts can be adapted to SDWSN, while others are very improbable. The network needs to be proactive and alert to any potential threats. Ali *et al.* [29] outline different network verification solutions in the traditional SDN networks, which deal with security configuration, threat detection, threat remediation and network verification (refer to Section 2.1.4). The programmability of the SDN makes it vulnerable to attacks [29], as applications can be installed quite easily.

With reference to WSN, the security solutions are mostly implemented on the sensor nodes and such protocols tend to be energy intensive and hence not practical [29]. These protocols can be implemented at the controller or application level. Some of these solutions are outlined in detail in [21]. The implementation of these concepts in SDWSN remains open to the research community. Furthermore, most security solutions can also be implemented through network function virtualization (NFV), which virtualises network functionalities.

From an SDN perspective, the control layer is more likely to be targeted by adversaries and therefore needs to be safeguarded [53]. The communication protocol also should be protected against any form of interception. The OpenFlow protocol uses a secure TCP protocol on port 6633 and/or also a secure channel based on TLS [39]. However, as Ali *et al.* [29] maintain, the impracticality of running SSL/TLS protocols on small devices, and the issue of securing network communication remain interesting research questions.

Another major challenge in traditional SDN networks is the fact that the switches are connected to the traffic-generating hosts, which could be used as gateways for attacks. However, in contrast, sensor nodes are at the periphery of the network, generating traffic. As a result, security threats through the sensor nodes are regarded as of less concern, as they are just sheer devices, which only understand a few entries in the flow table, although further study is needed to confirm this opinion. Table 2.6 maps the WSN OSI layers along the three planes of SDN, together with the security threats associated with those layers.

2.6.7 Distributed-control system

To realise scalability, reliability, and performance in SDWSN, an efficient distributed-control system is needed. Distributed control solutions have been proposed for SDN enterprise networks, but few for SDWSN. This underlines the need to investigate a novel distributed-control system for SDWSN without compromising any of the quality imperatives.

Table 2.6. Security threats on both WSN layers and SDN planes [1]

WSN OSI Layer	SDN Plane/Layer	Threat
Application	Application	Poor Authentication and Control Fraudulent flows rules insertion Poor access control and accountability Malicious Application DoS Northbound interface (API) attack
Transport Network	Control	Threats from applications DoS Unauthorised access Scalability and Unavailability Faulty or Malicious controller
Data Link Physical	Data	Unauthorised access Fraudulent rules Forged or False traffic flows Flooding, Spoofing Southbound Interface (API) attack Jamming, Tampering Sybil Compromised or hi-jacked controller Malicious node

2.7 DESIGN REQUIREMENTS

To address the challenges inherent in WSN, this section highlights some of the requirements that need to be considered in the future design of SDWSN. The requirements are depicted in Figure 2.9 which is an extension of the architecture presented in Figure 2.4. The main difference between the two is the fact that Figure 2.4 represents the requirements as currently applied by different studies, while Figure 2.9 enhances the requirements in accordance with the lessons learnt and future architectural considerations. The added functions are coloured dark blue. The relationship mapping between the controller and the sensor nodes is 1:M, however, this figure represents a distributed control environment.

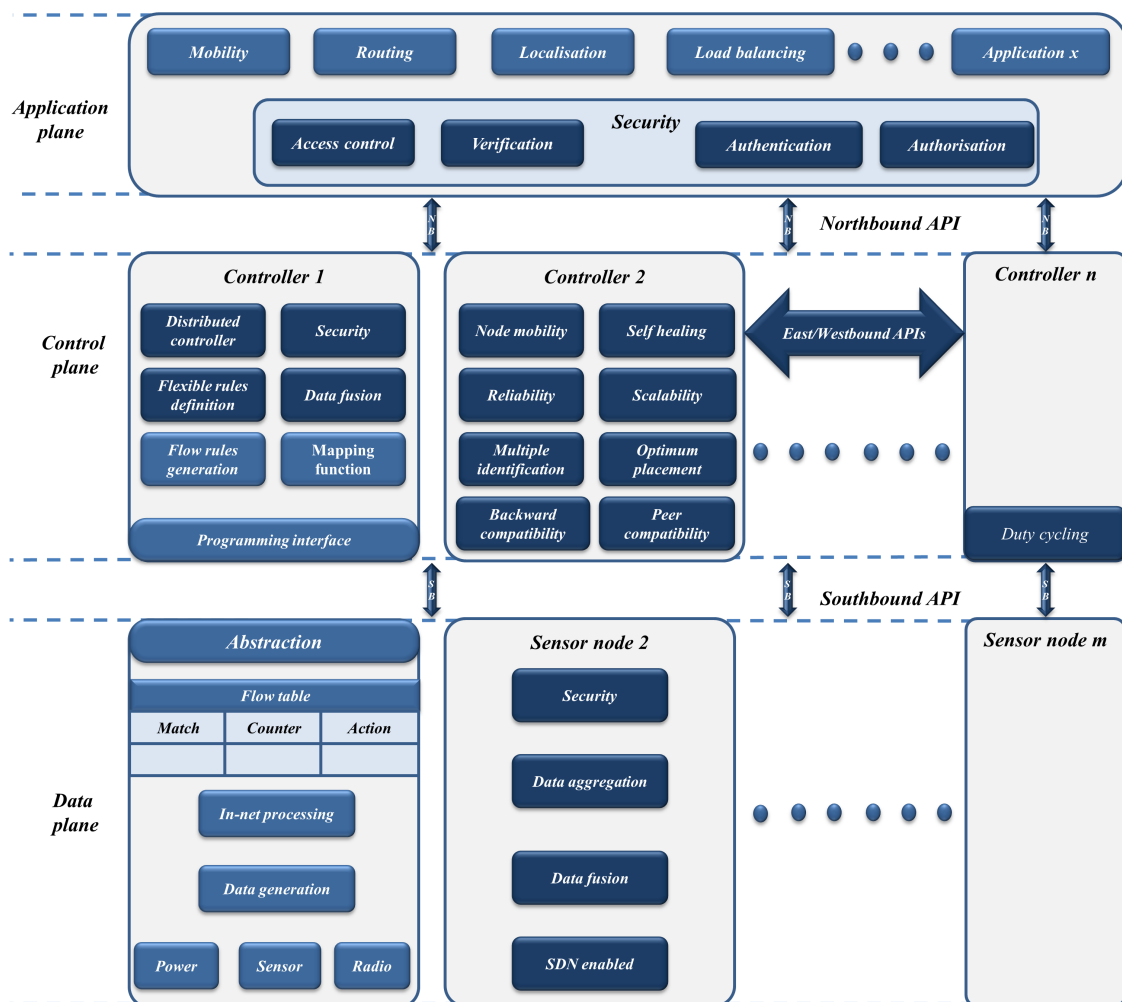


Figure 2.9. SDWSN design requirements. This figure captures the requirements as currently applied and also those that should be considered in future [1].

2.7.1 Duty cycles

The SDWSN should support duty cycling, i.e. switching the radio communication to a sleep mode, when not in use, Constanzo *et al.* [83]. That can be achieved in different ways, either reactively on demand or periodically through diligent synchronisation [21]. Low duty cycle operations are preferred in WSNs because high duty cycling could be more detrimental to energy efficiency. Saraswai *et al.* [175] evaluate this trade-off and conclude that energy consumption decreases if the duty cycle is less than 0.01% and 0.02% for dense and less dense networks respectively, but otherwise, duty increases energy consumption.

2.7.2 In-network data aggregation and decision fusion

In-network processing of data must be supported to avoid sending raw data to the sink or the controller [83]. Different data aggregation methods should be used to collate the data and transmit only the processed information. The aggregation of data could be based on the source, destination or the application attribute [21]. The protocol should allow a dynamic setting of these combinations through predefined permutations. The determination of what would guide the method of choosing these factors needs proper structural evaluation.

2.7.3 Flexible definition of rules

In line with the SDN premise of simple management, SDWSN should support flexible definition and application of rules and policies. There should also be a mechanism to prevent any advent clashes of rules or policies. FortNox [57] is a perfect example of such, although it is implemented for basic SDN, the inference is relant. The rule placement problem is expounded in a comprehensive review of different solutions in [173].

2.7.4 Node Mobility

Mobility is a very important design consideration for SDWSN, as the deployment of the sensor nodes varies according to application: some are structural, and others are not. Therefore, nodes could move

and change positions. The SDWSN should be able to deal with the inevitably rapid physical topology changes.

2.7.5 Unreliability of wireless links

WSNs consist of various RF (Radio Frequency) wireless communication links [75]. The wireless links are unstable and therefore not reliable. The instabilities are a result of common factors, such as limited bandwidth, node failures, etc. The duty cycling also affects the availability of nodes. The design of the SDWSN should consider the rapid topological changes caused by temporary unavailability of nodes.

2.7.6 Self-healing ability

Node failures in WSN should be expected and therefore the SDWSN needs to be dynamic with swift reorganisation to deal with such events. The controller is expected to be logically centralised to avoid a scenario of a single point of failure, i.e. physically distributed but operating logically as one controller. CPR recovery [176] is a technique used to ensure resilience in the event of controller failure. It restores the state of the applications (components) to the backup computer. This method employs a central controller and will therefore be subjected to scalability demands.

2.7.7 Backward and peer compatibility

The SDWSN should be compatible with existing WSNs, and SDN-based sensor nodes should be able to interface with normal sensor nodes. There should also be peer compatibility with other networks i.e. other OpenFlow networks. The SDWSN should also integrate well into the IoT framework and its protocols.

2.7.8 Data-centric and address-centric (multiple identification)

WSNs have always been considered data centric [21, 68], but recent studies are leaning towards address-centric approaches using IP addresses, particularly IPv6 (6LoWPAN) [73]. Moreover authors

in [71, 74, 135] state that augmenting WSNs with IP is an adequate solution for integrating WSNs into IoT. An SDWSN therefore should support both scenarios.

2.7.9 Scalability

The SDWSN framework should be scalable, and one way to realise that is to ensure that as the sensor nodes scale up, there is equally sufficient controller service. Although studies in [94, 139] state that one controller can handle millions of flows per second on traditional SDNs with switches, the same is yet to be ascertained for sensor nodes. De Gante *et al.* [27] state that to achieve reliable and robust scalability, the logically distributed controllers should also be physically distributed as in [88].

2.7.10 East/Westbound interface

To realise scalability, a distributed-control plane should be considered. A distributed-control plane requires consistent and synchronised communication between the controllers. The communication between the controllers is herein referred to as East/Westbound API [13]. As with the northbound API, very little attention has been paid to this interface. However, work such as SDNi [145, 177], which focuses on flow setup coordination and reachability of exchange information, is a progressive step. The development of distributed controllers such as Onix [178], Hyperflow [88], ONOS [87], Kandoo [141], Elasticon [94, 142], Pratyastha [93] and Aissioui *et al.* [58] will necessitate the development and standardisation of a common interface.

2.7.11 Northbound and Southbound interfaces

The northbound and southbound interfaces are very important to SDWSN for the fluidity of vertical cross-layer communication. However, much work was done which focused on the southbound interface and less on the northbound. As Nunes *et al.* [12] and Kim *et al.* [164] note, there is yet to be a standardised API or an interface for northbound communication. However, the SDWSN framework should create a platform for interoperability. The northbound interface was very important as it was connected to most of the functionalities that were removed from the node. This interface should be rigid with security and conflict prevention measures, [7, 108]. As heterogeneity is expected to be high in IoT, this interface could be used in conjunction with a virtualisation layer, same as OpenRoads [7] did

with heterogeneous wireless networks. Sneps-Snepp *et al.* [179] assert the need to have a metadata-based interface for the NB API and further suggest that the NB API will be based on REST [180]. However, unlike its counterpart service-oriented architecture (SOA) [181], REST does not offer metadata. Although SOA services such as WSDL (Web Service Definition Language), XSD (XML Schema Definition) provide metadata, they define different interfaces for each service application. Therefore, for the purpose of ubiquity and heterogeneity, REST would be feasible, but the metadata remains an open challenge.

2.7.12 Security

Security is a very important consideration for SDWSN frameworks. As SDWSN is envisaged to play a critical role in the IoT framework, it is equally imperative that security should be as stringent as possible. Most of the architectures do not have security as a built-in feature of SDN [53]; Kreutz *et al.* [13] rightfully state that security and dependability of SDN should be built in from design. Most of the current SDN security measures are earmarked for enterprise networks, where the network devices are switches or routers. These need to be adapted to consider wireless sensor network.

2.8 LESSONS LEARNT

This chapter reviews the current state-of-the-art application of SDN in WSNs, the SDWSN. The SDWSN falls within the broader context of the Internet of Things and as such, some of the related concepts have also been highlighted. This exercise has also been done to identify the purpose and role of SDWSN within the IoT space. The IoT paradigm seeks to create a networking environment for all devices expected to participate. Most of the devices will be equipped with sensors and actuators. Data from these devices will be carried by various networks such as enterprise, mobile wireless and optical networks. Various computing platforms such as cloud, fog, mobile cloud and mobile edge computing will feature prominently for service provisions. The application of SDN to these networks and computing platforms has been receiving significant and devoted attention from both academic and industrial research. This chapter has highlighted the bottom-up approach to the application of SDN to realise IoT. Sensors and actuators are located at the bottom as collectors of data, hence the evaluation of SDWSN.

WSNs are envisaged to be a significant building block of the IoT paradigm. However, WSNs inherently exhibit major constraints in terms of resources such as power, processing, memory, etc. To date, most of the attempted solutions have not been effective, which has exacerbated the challenges at cost. SDN offers a potential solution for some of these challenges. The SDN premise of decoupling the control logic from the forwarding engine brings a substantial respite as most of the energy-intensive functions are relocated to the controller. The chapter also explores the importance of SDN in WSNs. Issues such as energy, network management, configuration, scalability, routing, mobility, localisation interoperability, communication and security are envisaged to improve. It is also noted that the centralisation of the controller could pose a security risk, as it could be a potential target and a threat to reliability, resulting in a single point of failure. However, various research strides are being made to resolve this

The fusion of SDN and WSN begets SDWSN, which is a relatively research area. The chapter looks at different components of SDWSN, such as the architecture, network management, routing, etc. Due to the infancy of this field; most of the studies are still unravelling the architectural framework. Some architectures, such as that of Luo *et al.* [68] and [83] are more inclined towards the sensor node, while some such as those of Gante *et al.* [27] are inclined towards the controller. Some, such as those of Constanzo *et al.* [83] and Huang *et al.* [159], have some processing on the node while that of Jacobsson *et al.* [20] has a local controller on the node. Others, such as those of Constanzo *et al.* [83] and De Oliveira *et al.* [86], propose the use of a serial connection between the controller and the sink, which could stifle scalability. The lack of practicality is understandably common, with most of the work still on simulation, except in a few cases.

The challenges besetting SDWSN were presented for future consideration. These challenges include among others; some of the challenges inherent in WSN which are not wholly addressed by SDN, such as processing clarity, memory, etc. Other challenges identified include standardisation and security. Standardisation is key to the ideals of IoT heterogeneity, while security will also be central in IoT to ensure that future networks are secure and reliable. Finally, after the evaluation of the state-of-the-art architectures and their current challenges, design requirements were identified.

2.9 CONCLUSION

The SDWSN model is very challenging, as it comprises two unfledged models which are still entangled in their own complexities. The WSNs are resource constrained, which compels all research efforts to be energy conscious. Despite many efforts, this is yet to be fully realised and therefore have yet to reach their optimal efficiency. The introduction of SDN to WSN presents a very novel and progressive step in leveraging the challenges of resources in WSN. However, the SDN model brings along its own challenges, especially the trade-off between functionalities that need to be retained on the sensor device and the impact on common network aspects such as latency, congestion, etc.

This chapter has reviewed the current work on SDWSN. Most of the architectures proposed are still in the developmental stage. The prevalence of the Openflow protocol in SDN applications seems to have influenced even t the SDWSN model and therefore, we expect it to play a major role in the development of this model. Although there are still many challenges in this model, great strides have been made to date. However, the lack of standardisation in SDWSN is still a concern and standards should be developed to create oversight for compatibility and sustainability.

The chapter also discusses the SDWSN design requirements that need to be carefully evaluated and considered when designing and implementing a practical SDWSN framework. These design requirements would assist in overcoming the various challenges inherent to WSN, as well as the other challenges associated with SDN. Finally, the chapter tries to highlight some of the open research challenges that require more attention from the research community.

CHAPTER 3 SOFTWARE-DEFINED WIRELESS SENSOR NETWORK CONTROL SYSTEM

The controller plays a very important role in SDWSNs. As shown in Chapter 2, a central controller is not ideal for the efficacy of the network. However, the focus of most of the current research work is still on the architecture and early stages of development with few prototypes. Hence, the control systems are predominantly central. However, there has been much progress in distributed controllers for the enterprise SDN. Although not directly applicable to SDWSN; they present a cue that can influence the same for SDWSN. This chapter investigates and discusses the different variations of distributed SDN controller implementations. We discuss the current variations of distributed control systems from the traditional SDN perspective in Section 3.1 followed by current distributed controllers for the SDWSN in Section 3.2. Section 3.3 concludes the chapter.

3.1 SDN CONTROLLER IMPLEMENTATIONS

There are different implementations of the controllers, such as centralised, distributed, logically centralised but physically distributed and data plane extensions. This section reviews the current control systems in SDN from the perspective of SDWSN for viability purposes.

3.1.1 A single centralised controller

A centralised controller is an epitome of the basic SDN implementation where a single central controller controls the entire network. An example of a centralised controller is depicted in Figure 3.1. SDN

central controllers include Floodlight [140, 182], Ryu [183], Maestro [138], NOX [56], Beacon [139]. Major concerns with a single central controller are scalability, reliability, and congestion. SDWSNs are envisaged to grow rapidly and to operate in heterogeneous environments; a central controller will stifle the growth and flexibility of the network. As the network scales, the distance between the sensor nodes and the controller could result in a deterioration of performance, and service degradation. Also, sending the data in one direction might cause excessive overheads and affect the validity of the data carried. SDWSN carries data that changes rapidly, any delay might delegitimise the data or render it irrelevant.

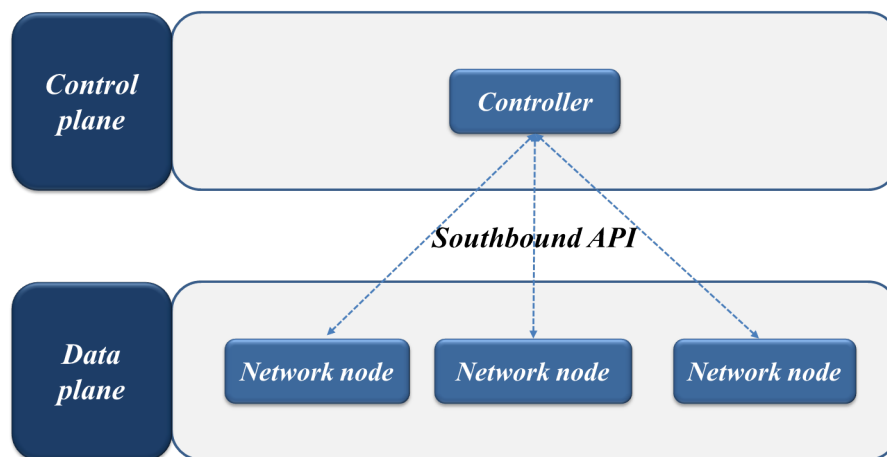


Figure 3.1. SDN central controller [1].

3.1.2 Distributed controllers

To overcome the drawbacks of the single centralised controller mentioned above, several researchers used multiple distributed controllers. A distributed-control system is depicted in Figure 3.2. In TinySDN [86] multiple controllers are used. The prototype consists of a sensor node and a sink node attached serially to a controller. Olivier *et al.* [144] propose a cluster-based SDWSN architecture which also has a base station. They apply the SDN model in a clustered WSN for the formation of what they refer to as a software-defined cluster sensor network. The sensor network is organised in clusters (SDN domains) comprising a simple node (sensor) and a gateway or cluster head. The SDN cluster head controls and coordinates all sensor nodes in its domain.

SDNi protocol [177] is an East/Westbound interface between controllers in a distributed-control environment referred to as controller domains. A domain is a network cluster with a controller.

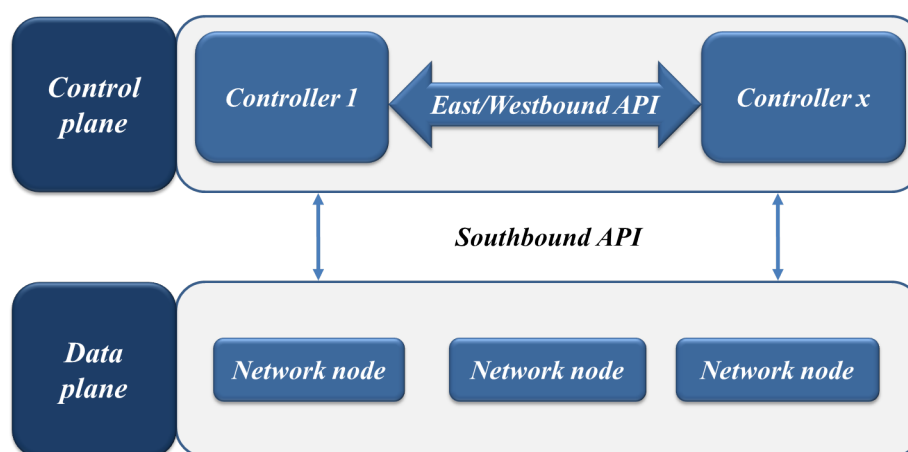


Figure 3.2. SDN distributed controllers [1].

SDNi's main purpose is to coordinate controller behaviours and to facilitate the exchange of control and application information across multiple domains. The protocol has three types of message: reachability update which exchanges reachability information to facilitate inter-SDN domain routing, flow setup/tear-down/update request which coordinates the flow setup, and capability update which exchanges network-related capabilities such as QoS in the domain.

3.1.3 Logically centralised but physically distributed

Logically centralised but physically distributed controllers operate as a central controller though they are physically apart, as shown in Figure 3.3. Hyperflow [88] is an event-based logically centralised but physically distributed controller for OpenFlow. It is based on NOX [56], a network operating system controller; built as an application. The Hyperflow application resides in each NOX controller in the network. The application propagates network events across the controllers, thereby synchronizing their network views. Hyperflow uses publish/subscribe messaging to facilitate the lateral controller communication. The published events are stored using WheelFS [184], a distributed-file system. Switches are connected to a controller close to them and upon controller failure, they must be reconfigured.

ONOS [87] is a distributed-control architecture based on Floodlight [140] which aims to ensure scalability, performance and availability. It is also installed into the physically distributed controllers which operate as a unit. Each node has a global view of the network. The ONOS instances control a subset of switches. The ONOS instances propagate state changes of the switches to the rest of the

controller instances in the cluster. A switch connects to multiple ONOS instances but only one can be the master. Upon failure, the switches elect a new master. ONOS uses a distributed data store for the multiple clusters. OpenDaylight (ODL) [185, 186] is another distributed SDN controller based on Beacon which like ONOS, employs a multiple cluster of controllers maintained through a distributed data store. ODL uses model-driven software engineering (MDSE) for inter-model relationship and model-driven network management for interprotocol configuration management such as NETCONFIG and RESTCONFIG [186]. Both ODL and ONOS are based on the OSGI framework.

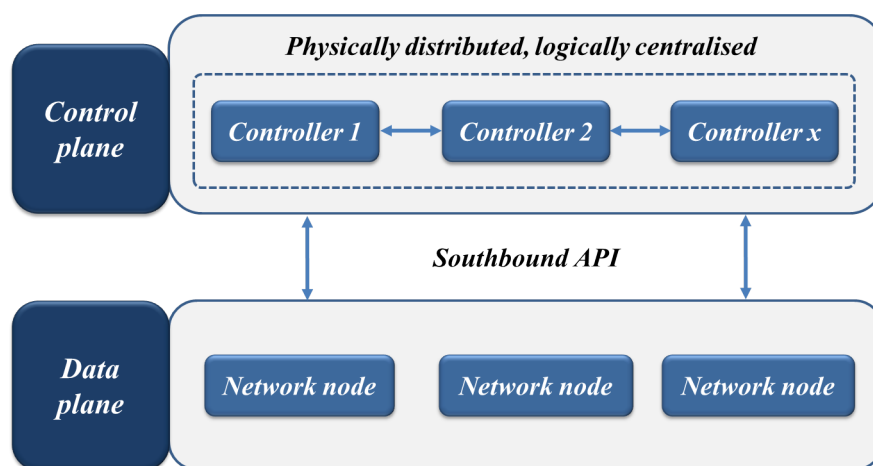


Figure 3.3. SDN logically centralised but physically distributed controllers [1].

Disco [92] is a distributed-control system for WAN and overlay networks. The authors aim to cater for heterogeneous and constrained networks as they maintain that current solutions are not adaptable. Disco is based on a Floodlight [140] controller, built as an application. It defines two types of communication: interdomain and intradomain. The intradomain communication monitors network events within the domain cluster and uses the network state to do flow prioritisation. The interdomain communication allows SDN controllers to exchange aggregated network-wide information between each other. The interdomain communication employs two methods; messenger, which ensures connectivity by using the advanced message-queuing protocol (AMQP) and agents, which exchange the actual information. The communication supports diffusion, flooding, and publish/subscribe messages.

3.1.4 A data plane extension control

Other solutions, referred to as data plane extension control systems, enhance the data plane with more control functionality to reduce the overhead towards the controller; Figure 3.4 depicts the architecture.

Difane [89] is a data plane extension distributed-control architecture which keeps most of the traffic on the data plane. Difane offloads some control functionalities to the data plane switches referred to as authority switches which have more power and processing resources. The controller generates flow rules, and then passes them on the authority switches which then install them to the rest of the switches. In Devoflow [90], the authors aim to devolve the control of most flows to the switches as well. The controller would still control a few significant flows referred to as elephant flows; these are the flows that carry heavy traffic. Some of these switches can even make routing decisions.

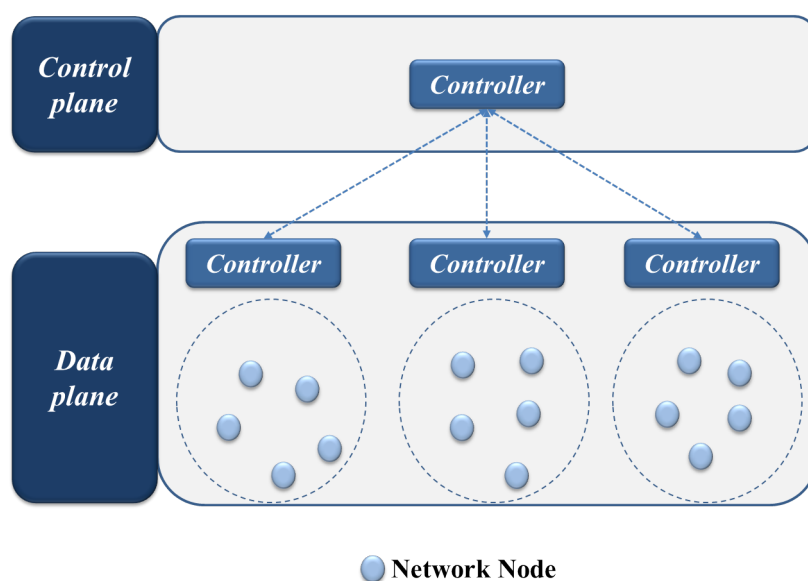


Figure 3.4. SDN data plane extension controller [1].

Kandoo [91] is another data plane extension solution which employs a rather different approach or method as compared to Devoflow and Difane. Kandoo scales the control plane with local controllers at the data plane, thereby creating a two-level control architecture. Local controllers execute local applications, but have no interconnection amongst themselves and no global network view. The root controller is logically centralised and has a global view of the entire network. It executes nonlocal applications, i.e. applications that need global network knowledge. The root controller, in turn, controls the local controllers. Local controllers could also be implemented at the switch. Frequent events are processed at the data plane while rare events are controlled at the central control. The root controller must subscribe to events from the local controller; otherwise those events would not be propagated.

3.1.5 Switch to controller mapping configurations

Most of the distributed solutions referred to above use static configuration to map the switches to the controllers. Elastic solutions in contrast, offer more flexibility and dynamic assignment of switches to controllers [58, 93, 94, 142]. Elasticon [142] is a distributed-control architecture in which switches are dynamically shifted across the controllers to balance the load; the controller pool is also dynamically shrunk or increased. The controllers have a synchronised coordinator providing a consistent state of control of the network. A switch connects to many controllers but only one can be the master while the others are slaves. The load on the network is shared. There is a threshold that is used to migrate switches to other controllers and to reduce or increase the controller pool. It has a load adaptation algorithm and migration protocol. The model was tested using an enhanced Mininet simulator.

Pratyaatsha [93] is also an elastic control solution which, like Elasticon, dynamically assigns SDN switches. In addition, it also assigns SDN application partitions to the distributed controllers. It aims to address two issues: minimising flow setup latency and minimising costs through efficient resource allocation, which includes memory as opposed to only CPU load. Pratyaatsha uses the integer linear programming (ILP) algorithm to assign application state partitions and switches effectively to controllers. Aissioui *et al.* [58] propose another elastic solution for 5G mobile cloud management. The authors aim to address issues of scalability and performance in the context of 5G networks. The solution is based on mobile cloud computing, particularly the follow-me cloud (FMC) concept which ensures that mobile users, subjected to many movement constraints and migrations, are always connected to an optimal data centre. The proposed elastic control solution has two levels: global FMC controller (G-FMCC) and local FMC controller (L-FMCC). The L-FMCC is deployed on demand by using Network Function Virtualisation (NFV) depending on the network dynamics and traffic patterns. G-FMCCs are permanent and responsible for generating, managing and installing OpenFlow rules which ensure seamless migration of services on the cloud.

3.1.6 Summary of discussion

The different types of controller implementation are summarised in Table 3.1 below, which includes most of the existing distributed systems in addition to a few centralised systems. The SDWSN controllers are mostly centralised since most of the work is still in a development stage. In contrast, distributed

controllers are yet to be implemented in SDWSN. However, traditional SDN-distributed controllers can be used as a reference point to develop a tailored distributed controller for SDWSN.

All controller types, apart from data plane extension, keep the control functionality at the control plane, while the data plane extension devolves some of the control functionality to the data plane to reduce the overhead on the centralised controller and reduce traffic in the network. In the logically centralised but physically distributed types, an east/westbound API is used to enable interaction amongst the controllers. However, not all distribution systems consider the use of the east/westbound API. Table 3.2 below summarises the details of the controller type implementation and further highlights the advantages and disadvantages of each type.

Table 3.1. Distributed controllers in SDN [1].

Controller	Implementation				Switch mapping		SDWSN	SDN
	Centralised	Distributed	Logically Distributed	Data Plane Extension	Static	Dynamic		
TinySDN [86]		✓				✓	✓	
SDWN [83]	✓				✓		✓	
SDCSN [144]		✓			✓		✓	
Cognitive SDWSN [159]	✓				✓		✓	
Hyperflow [88]			✓		✓			✓
ONOS [87]			✓		✓			✓
Disco [92]			✓		✓			✓
Difane [89]				✓	✓			✓
DevoFlow [90]				✓	✓			✓
Kandoo [91]				✓	✓			✓
Elasticon [94, 142]		✓				✓		✓
Pratyatsha [93]		✓				✓		✓
Aissioui <i>et al.</i> [58]		✓				✓		✓

3.2 SDWSN CONTROLLER

TinySDN [86] is one of the earliest distributed-control solutions for SDWSN where multiple controllers are used with the aim of reducing control traffic. It consists of a sensor node and a sink node which are connected to a controller on a serial port. The SDN sensor node must first find a controller to belong to by using the collection tree protocol (CTP). This solution was tested by using two controllers.

Table 3.2. Detailed comparison of the controllers [1]

Controller Type	SDN Plane	SDN API	Advantage	Disadvantage
Centralised	Control	Northbound Southbound	Global network view Informed decision	Central point of failure Congestion Not scalable Performance degradation
Distributed Logically distributed	Control	Northbound Southbound East/westbound	Scalability Quick response Ad hoc control access Improves security	High costs Static configuration Unbalanced load distribution High memory cost Synchronisation of large data
Data Plane Extension	Control Data	Northbound Southbound	Reduces overhead Quick response time Improves performance Improves security	Increases cost Depends on central controller

The serial communication between the sink and the controller limits the controller and the sink to a one-on-one relationship. This poses a problem of scalability. These architectures will only be viable in a small controllable network. The fact that sensor nodes keep track of their neighbours, together with the flow table, might consume space, which is scarce. The nodes also have to make some routing decisions; this task should be handled by the controller via the flow tables. The authors extended this work in [187] where they considered the use of a spot-led architecture for the controllers. This architecture is both distributed and hierarchical; it consists of local nodes in a distributed format, as well as a global controller overseeing all the local controllers. The spot-led controllers were tested as standalones.

SDN-WISE [136] is a comprehensive SDWSN framework based on state automata. The SDN-WISE software stack gives a detailed description of the SDN sink nodes and sensor nodes. The stateful structure includes the flow table composition in the flow of Openflow. It also details the protocol architecture dealing with the packet handling and processing, and topology discovery techniques. The stack is adaptable to various SDN controllers and therefore an adaptation with ONOS was implemented by the authors and tested for interoperability with the enterprise network [188].

The combination of SDN-WISE and ONOS is progressively in the right direction towards the realisation

of an effective and efficient SDWSN for IoT. In [189], this solution was tested by using multiple controllers in a distributed fashion. The authors of SDN-WISE also implemented a custom Java controller to test this framework which used Dijkstra's algorithm. Dijkstra's algorithm finds the shortest paths between nodes in a connected graph.

3.3 CONCLUSION

The SDN model centralises the control intelligence of the whole network; although this abstraction brings many positive benefits, it also introduces several challenges. The entire network could be at risk if the central controller becomes compromised. Failure of the controller could also negate the availability of the network, thereby rendering it unreliable. The performance of the network will also suffer due to the overhead. The distance between the gateway nodes and the controller would also affect the performance. Therefore, a centralised controller would not be viable for a wireless sensor network, especially in view of the inherent challenges such as unreliable links, low bandwidth etc. Performance and efficiency will also suffer as the network grows.

To realise scalability, reliability, and performance in SDWSN, an efficient distributed-control system is needed. Distributed control solutions have been proposed for SDN enterprise networks but few for SDWSN. This encourages the need to investigate a novel distributed-control system for SDWSN without compromising any of the quality imperatives.

CHAPTER 4 FRAGMENTATION-BASED DISTRIBUTED CONTROL SYSTEM FOR SOFTWARE-DEFINED WIRELESS SENSOR NETWORKS

This chapter discusses the proposed system model in detail. The purpose of this research work is to use fragmentation [8]² as a method of distributing the SDWSN control logic with the aim of achieving reliability, performance, and scalability. We start by providing a brief overview of distribution and the rationale behind the proposed model in Section 4.1. This is followed by a brief background of distributed systems and their relation to database systems in Subsection 4.1.1. The method of fragmentation is then discussed in detail in Subsection 4.1.2. In Section 4.2, we discuss the gossip protocols which are the main building blocks of the proposed model. These are best-effort and anti-entropy. In Subsection 4.2.1, we discuss the way in which these two gossip protocol algorithms are optimised to realise fragmentation. We also discuss the time complexities of these two algorithms before and after optimisation in Subsection 4.2.2. Section 4.3 concludes the chapter.

4.1 DISTRIBUTION

Distributed-control systems have been around for some time; they only change in form, scope, and application. Therefore the rationale behind a distributed application/computing is commonly applicable

² ©2017 IEEE. Reprinted, with permission, from Kobo H.I., Abu-Mafhouz A.M., Hancke G.P., Fragmentation-Based Distributed Control System for Software-Defined Wireless Sensor Networks, IEEE Transaction on Industrial Informatics, May 2018.

across the spectra of applications. However, there are traits that are unique and exclusive to a particular form or application of distribution.

The SDN's centralisation of the control logic necessitates the need for distribution. The fundamental reasons for distributing the control logic are mainly to avert a central point of failure (addressing reliability), congestion, delay, scalability and load balancing. Centralisation essentially means that the whole network relies on the controller for functionality and, if compromised, the whole network will also cease to function. Also, it becomes a potential target for malicious attacks; to the detriment of the whole network.

The rationale behind distributed-control for SDWSN is based on these facts and furthermore, it also addresses WSN's unique challenges. Unlike enterprise SDN networks, SDWSN have inherent limitations of energy, memory, data rate etc. Hence, the distribution criterion of the SDWSN should take all these factors into consideration. Another distinctive feature of SDWSNs is the fact that they are used to capture real-time events which change rapidly and are therefore delay sensitive. The sensed data could be rendered redundant if it does not arrive at the controller on time. The delay could be due to various factors such as the distance between the sensor nodes and the controller, the limited data rate and the frequency of the data transmission. The distributed control is ideally suited for such scenarios as it shortens the distance to the controller, shares the load and ensures faster response times.

4.1.1 Background

It is a well-known fact that distributed systems found more prevalence in database theory. E Brewer profoundly stated at the 2000 Symposium of distributed computing that "*in any highly distributed data system there are three common desirable properties: consistency, availability and partition tolerance. However, it is impossible for a system to provide all three properties at the same time.*" [190, 191]. This became known as the CAP theorem. Coronel and Morris [191] define these three properties as follows [8]:

- Consistency – All nodes should see the same data at the same time, which means that the replicas should be immediately updated.
- Availability – A request is always fulfilled by the system.

- Partition tolerance – The system continues to operate even in the event of a node failure.

In the context of this research the CAP theorem would mean:

- Consistency – all fragments (local controllers) have the same network state all the time i.e. symmetric.
- Availability – all nodes are available.
- Partition tolerance – the network continues to function after a node failure.

Another phenomenon common in database systems is ACID (Atomicity, Consistency, Isolation, and Durability); regarded as the four properties of a transactional database. ACID ensures that all transactions result in a consistent database state. Coronel and Morris note that this is well suited to centralised and small distributed database systems [191]. Otherwise, latency becomes an issue as the system scales. They further state that it is for this reason that many systems sacrifice consistency and isolation for availability, which leads to another phenomenon, BASE (basically available, soft state, eventually consistent). In BASE, data exchanges are not immediate but propagate slowly until all nodes are eventually consistent [190].

Distributed SDN control systems draw reference from the above, particularly the database systems. However, the fundamental roles of an SDN controller and a database system are distinguishable. The primary task of a database is to store data and enable the CRUD (create, read, update and delete) operation. The SDN controller, on the other hand, is an engine of the network; more pointedly to control the infrastructure devices by defining data propagation rules. Furthermore, SDWSN introduces another dimension, different from the traditional SDN by putting the sensor nodes at the periphery of the network. This also tilts the paradigm of the controller's role. It is therefore against this background that our system follows BASE as a consistency model, thereby preferring availability over consistency.

This research work undertakes to provide an eventual consistency model and to customise it for SDWSN control. This process takes cognisance of the perpetual SDWSN challenges such as limited energy, bandwidth etc. This method applies concurrency and parallelism to the local controller nodes. This effectively ensures that each local controller node independently controls its segment (cluster) of the network and work concurrently with the other local controller nodes. These nodes would operate

simultaneously, thus guaranteeing parallelism. This would allow a faster response between the sensor nodes and the controller. Most of the data would be upstream and only infrequently would the controller send control instructions to the devices.

The eventual consistency data model falls within the classification of BASE which has a weaker consistency but high availability and performance. It is also referred to as optimistic replication or lazy replication which, unlike pessimistic replication (symmetric), enhances concurrency and parallelism. Concurrency enables unit processes to be executed independently while parallelism enables simultaneous execution of processes. The eventual consistency is, however, still applied, albeit not laterally but vertically to the global controller node. The upward propagation of the cluster states to the global controller does not affect the operation of the network.

4.1.2 Fragmentation

The concept of fragmentation is prevalent in distributed database management solutions. There are different ways of doing fragmentation such as through relations in relational databases or classes in object oriented database [192]. Therefore, any attribute of a subject object can be used in fragmentation for a particular purpose. Khan *et al.* [192] summarises the main reasons of database fragmentation as follows [8]:

- Increases locality of reference of queries to the database.
- Improves reliability and availability of data,
- Improves the performance of the system.
- Balances storage capacities.
- Minimises communication costs among sites.

These reasons apply across the whole spectrum of fragmentation. Ciriani *et al.* [193] propose a method of using fragmentation and encryption to protect the confidentiality of data in sensitive distributed databases such as medical information etc.

We propose fragmentation for our distributed-control solution. Fragmentation entails dedicating part of the control system to different segments (fragments) of the sensor network. In addition to fragmenting

the control of the network, this could be extended to abstracting different sensor traits together e.g. temperature nodes. Figure 4.1 depicts the distributed-control system with fragmentation. This architecture is akin to that of Kandoo [91], which also has local controllers as well as a global controller overseeing the whole network. However, Kandoo is an SDN controller dedicated to traditional SDN networks and does not implement fragmentation as defined in this research work.

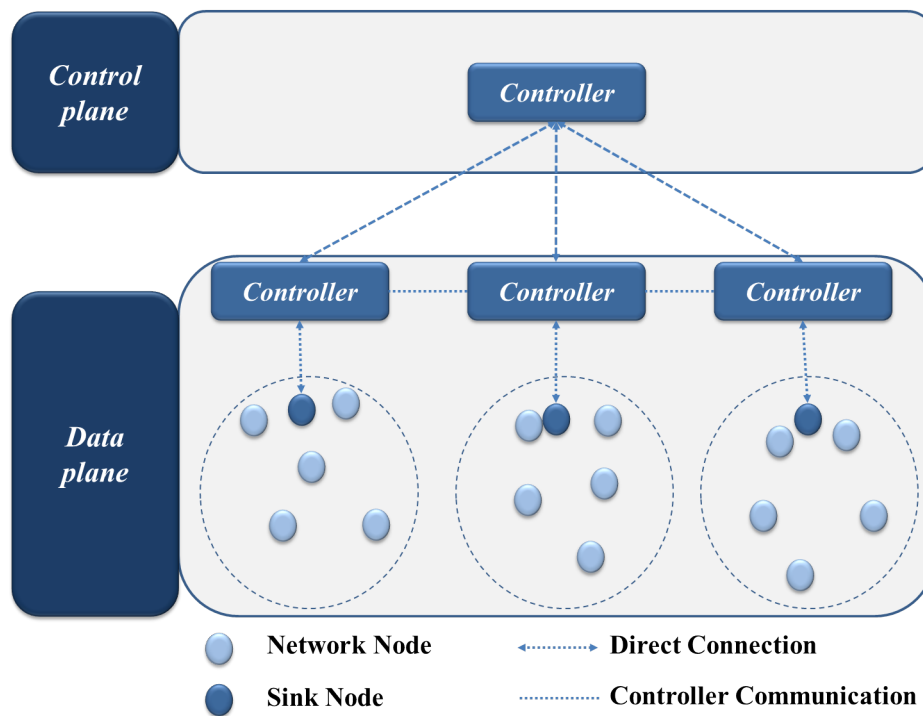


Figure 4.1. Distributed-control system for SDWSN with fragmentation [8].

Fragmentation takes distribution further by taking the control logic closer to the infrastructure devices. Although the communication between the sink node and the controller is through IP protocol, the sink nodes still possess less resources and therefore, placing the controller close to the sink node saves a considerable amount of power. This also relieves the sink node from pressure because of its low data rates or bandwidth. This method involves a two-level control structure where there is a global controller overseeing the whole network and a local controller in charge of only a portion or segment of the network. The local controllers only have knowledge of the portion of the network they are controlling. The characteristics of each controller are as follows [8]:

1. Global controller

- Global view/knowledge of the network.
- Load balancing.
- SDN functionalities of a controller.
- Failure mechanism.
 - Failure of the global controller does not affect the operation of the network, only a temporal disruption of display and other functions that require global knowledge.
 - Replication method used to create redundancy.

2. Local Controller

- Takes charge/control of a cluster.
- Updates the global controller.
- Has local cluster knowledge.
- Lightweight for cost effectiveness.
- Failure mechanism.
 - Another local controller takes over.
 - Learns the cluster state from the central controller.
 - Sink connects to the closest controller.

3. Sink Node

- Connects to the local controller (the closest).
- Communicates with the sensor nodes.
- Relays/conveys information to the local controller.
- Uses RF to communicate with the other sensor nodes and internet to connect to the local controller.
- Failure mechanism.
 - Sensor nodes find another sink within their reach.

There are three fundamental things that distribution seeks to achieve: reliability, scalability, and efficiency. However, special consideration should be accorded in SDWSN due to the inherent low capacity of WSNs. Therefore, in addition to the generic reasons of fragmentation as stated above, the reasons for SDWSN control fragmentation are [8]:

- Dedicates controller to a cluster.
- Avoids having global knowledge on all controllers, reduce overhead cost by updates.
- Improves responsiveness.
- Allows proper isolation of sensed data types.
- Reduces redundancy on the local nodes.
- Updates to the global controller cannot affect the operation of the network i.e. delay, congestion.
- Keeps latency low by – keeping control close to the sensor nodes.

Figure 4.2 depicts the high-level design of the research mechanism technique; each controller, herein referred to as a fragment controlling a cluster of nodes. This diagram shows all concepts considered in this chapter in order of application.

4.2 EPIDEMIC/GOSSIP PROTOCOLS

An epidemic is the spread of disease (infectious) to a large number of people in a population within a short time [194, 195]; whilst Gossip is the spread of rumours in an informal way in social circles. These two concepts, with the common denominator of “spread”, informs the basis of Gossip/Epidemic protocols. Gossip protocols disseminate information across a distributed system by using a gossip-like method. A participant randomly pairs with a peer and exchanges update information between them and after a time, full consistency is reached.

Epidemic algorithms feature prominently in information dissemination solutions in large systems as they offer much scalability, ease of use or deployment, robustness, and resilience [196]. Recently, the adoption of these algorithms has spread beyond information dissemination to other platforms such as failure detection, data aggregation, network management, load balancing, synchronisation, discovery and monitoring, and database replication [197–204].

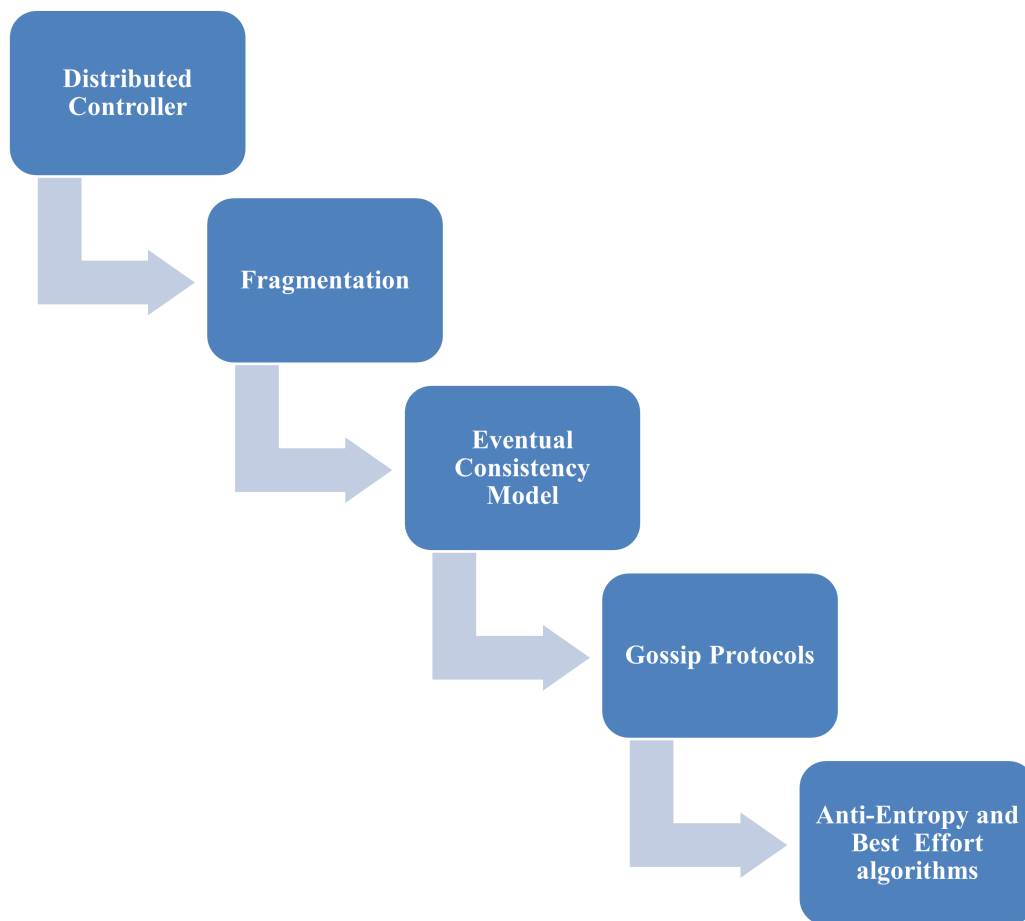


Figure 4.2. The complete research structure [8].

Montresor [200] defines the generic characteristics of gossip protocols or factors that a gossip protocol should at least satisfy:

- Random peer selection or guarantee of peer diversity.
- Only local information available at all nodes.
- Periodic communication.
- Limited transmission and processing capacity per round.
- All nodes use the same protocol.

At the centre of gossip algorithms is random peer selection called peer-sampling service. This service is very important as it provides the algorithms with the participating nodes [201]. The peer-sampling

service is simple but crucial as it makes a huge contribution to the overall performance of the algorithm. Jelasiy *et al.* [201] discuss different ways of implementing peer sampling.

The evolution of gossip protocols was initially introduced by Demer's 1987 paper titled "Epidemic algorithms for replicated database maintenance" [205]. There are different gossip protocols in Best Effort also called direct mail, Anti-entropy and Rumour Mongering. This proposed system focuses on Best Effort and Anti-entropy.

Best Effort ensures that every new event or update is sent immediately to all other nodes. Anti-entropy compares the replicas/nodes and reconciles the differences; thus updating each copy to the latest one [206]. Rumour-mongering floods the network with updates for a time sufficient enough to have all nodes updated. These methods suit networks with a moderate latency tolerance, however, they would potentially lead to latency challenges under high update loads [206]. The SDWSN is envisaged to be a high update network with little to no latency tolerance and, given the low capacity bandwidth; these two methods could be problematic.

Best Effort or direct mail is event based. An update to other nodes is triggered by the occurrence of an event. Upon receiving an event, the node will broadcast that event to all other nodes in the cluster. An event could include any occurrence of an update, such as a new node, node failure etc. The receiving nodes in the cluster evaluate the recency of the event. If recent, the matching entry will be updated accordingly. Otherwise, the sender will have to update its state. The following tables show the pseudocodes of the algorithms and their time efficiency. Algorithm 1 shows the constructor which forms the base of all the algorithms discussed. Algorithm 2 shows the pseudocode of the Best Effort algorithm. The time complexity of the algorithm is $O(n) + O(1)$ which is $O(n)$.

Algorithm 1 Constructor [8]

Require: Let S be a set of participants: $S = \{p, q, r, \dots\}$

The state of the participant p is modelled as state: $K \leftarrow V.T$

K is the set of Keys

V is the set of values

T is the set of Timestamps

Therefore the state $s(k) \leftarrow (v, t)$

Algorithm 2 Best Effort algorithm [8]

Require: Let S be a set of participants: $S = \{p, q, r, \dots\}$

Upon receiving event: $p.state(v, t_i)$ $O(n)$

for $q \in S$ **do**

Send state to q

end for

Upon receiving state: (v, t_i) $O(1)$

if $state.time < t$ **then**

$state \leftarrow (v, t)$

else

$(v, t) \leftarrow (v, t_i)$

end if

Total time complexity $O(n) + O(1)$ $O(n)$

The Anti-entropy algorithm is almost similar to Best Effort in content but different. Anti-entropy is periodic, therefore the synchronisation occurs after every set period. A peer periodically chooses a random partner from a list of peers (nodes) and starts to exchange state information. Thus, peer p sends its state to q , and q applies it to its own state, this is called the push method. Otherwise in the pull method, p sends its state to q which only consists of keys and timestamps, then q responds with appropriate matching updates to p . The pull-push method is the combination of both methods, while q sends updates to p as in the pull method, it also sends its outdated values compared to p . This is the most used and most efficient method. By using the pull-push method, a node sends its state to a peer node. The peer node checks the state received for recency, and if recent, it updates its state accordingly. Otherwise it sends a message to the sender node with a set of updates. Upon receiving the reply, the original sender (now the receiver) applies the changes, first by checking the recency, then by updating if recent. If an entry is missing, it is requested. The pseudocode of the algorithm is described in Algorithm 3. The total time complexity of this algorithm is $O(n \log n) + O(1) + O(1)$, which equals $O(n \log n)$.

Algorithm 3 Anti-entropy algorithm [8, 205, 206]

Require: Let S be a set of participants: $S = \{p, q, r, \dots\}$

for every T period **do**

Randomly select a peer q from set of peers S $O(n \log n)$

Send state: (v, t_i) to q with state (v, t) : Update

end for

Upon receiving $p.state$ $O(1)$

if $p.state.time > q.state.time$ i.e t **then**

$q.state : (v, t) \leftarrow p.state : (v, t_i)$

else if $p.state.time : (v, t_i) < q.state.time : (v, t)$ **then**

send a reply to p

end if

Upon receiving a reply $O(1)$

if $p.state.time < q.state.time$ **then**

$p.state : (v, t_i) \leftarrow q.state.time : (v, t)$

end if

if $p.state : (v, t) \in S$ such that $(v_i, t_j) \notin q$ **then**

request (v_i, t_j)

end if

Total time complexity $O(n \log n) + O(1) + O(1)$ $O(n \log n)$

The ONOS architecture is based on the eventual consistency data model; however, applications that require stronger data guarantees can use the strong consistency model as an alternative. The strong consistency model is backed by the RAFT [24] algorithm. The eventual consistency uses Best Effort to update all other peers when an event occurs and Anti-entropy to resolve the differences amongst nodes. The proposed method uses the eventual consistency model. At this stage, the architecture allows a two-level control structure with eventual consistency which is backed by the Best Effort and Anti-entropy algorithms. This architecture enables the network to scale, which satisfies the scalability objective of this research study. This mimics the Kandoo [141] architecture discussed in Chapter 3. In addition to scalability, this architecture takes controller functionalities closer to the infrastructure devices of the network. Like Kandoo [141], this method suits networks which are resourceful. Each local controller node has a global view of the network as enabled by the synchronisation of the Best

Effort and Anti-entropy algorithms. However, this structure does not suit the SDWSN, because of its unique characteristics as outlined above. It also does not improve the efficiency and performance of the network. Although it improves scalability, it increases the capital costs of the network because the size of the controllers would have to be the same size and specification for optimal functionality. Otherwise, the local controllers, if undersized, could be overwhelmed by the colossal amount of data that it does not only carry but also process. Secondly, the local controllers process so much data which they do not utilise. Sensory data is rapid, live, and mostly unidirectional and when the other controller nodes reach convergence (eventual consistency updates), the data could already be irrelevant. Thus, resources are not used efficiently. Thirdly, this has a direct impact on the performance and lastly, the two-level structure would add an overhead which does not have a positive impact overall. Therefore, although the structure brings about scalability, it does not achieve efficiency and improve performance and therefore it requires optimisation.

4.2.1 Optimisation

Having achieved scalability by adding local controllers at the edge of the network, optimisation is sought to achieve efficiency and performance. This optimisation should also reduce the capital costs as identified above. The first intervention towards fragmentation is to change the behaviour of the gossip algorithms. The two algorithms stated do not bring the desired outcome of fragmentation. Therefore, the two algorithms are redesigned to ensure fragmentation. The new algorithms will be referred to as Best Effort with fragmentation and Anti-entropy with fragmentation to retain consistency.

The Best effort algorithm is redesigned to ensure that it only sends updates to the global controller, thus all updates triggered by events are sent to the global controller. Upon receiving an event, the node sends it to the global controller. This step reduces the computation of the algorithm from $O(n)$ to $O(1)$. The global controller then checks if the event is recent before updating its state. If the global controller does not have that entry, it is created; however, if an entry exist in the global controller and not in the local controller, the local controller ignores the entry. This is to ensure that each local controller has a view and control of its cluster of the network. The Best effort algorithm is modelled as described by the pseudocode in Algorithm 4 for the fragmentation model. The steps described in Algorithm 4 are further depicted by the flowchart in Figure 4.3. The flowchart is generic and caters for all updates from the global and local controllers. The peer sampling has been simplified because it is now between

the local controllers to the global controllers in a many-to-one mapping (M:1) and from the global controller to the local controllers in one-to-many mapping. Redundancy of the global controller occurs through replication.

Algorithm 4 Best Effort algorithm with Fragmentation [8]

Require: Let S be a set of participants: $S = \{p, q, r, \dots\}$

Upon receiving event: $p.state(v, t_i)$ $O(1)$

Send state to global controller gc: (v, t)

Upon receiving state: (v_i, t_j) $O(1)$

if $p.state.time < gc.state.time$ **then**

$p.state \leftarrow gc.state : (v, t)$

else

$(v, t_i) \leftarrow gc.state : (v, t)$

end if

if $p.state : (v, t) \in S$ such that $(v_i, t_j) \notin q$ **then**

$O(1)$

if sender is global controller **then**

ignore

else

request (v_i, t_j)

end if

end if

Total time complexity $O(1) + O(1) + O(1)$

$O(1)$

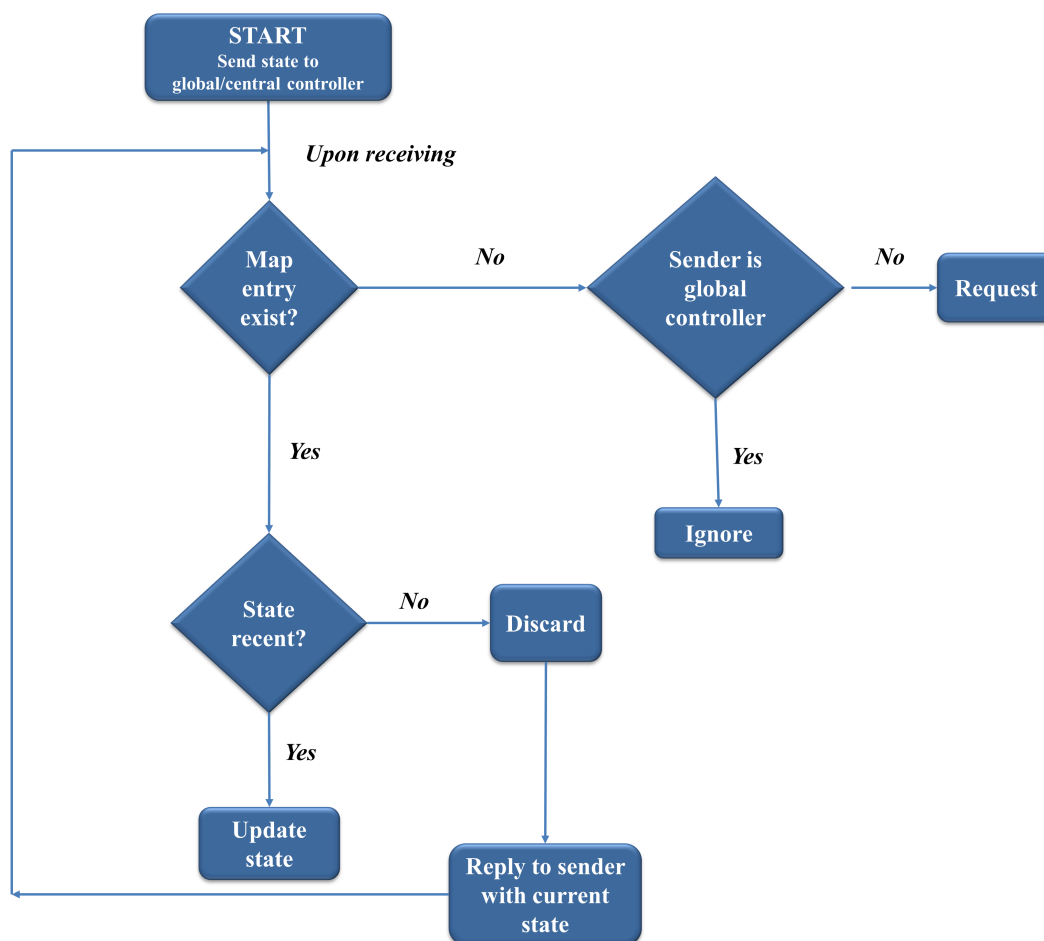


Figure 4.3. The flowchart of the Best Effort algorithm with fragmentation [8].

The Anti-entropy algorithm is also redesigned from updating its peers to only updating the main global controller. The Anti-entropy-handling method is enhanced to be able to distinctively handle updates from either the global controller or the local controller. The local controller executes the main updates to the global controller. If the local controller has new devices, the global controller will be updated during the data exchange. However, the local controller cannot accept any new devices from the global controller (might be from other clusters). The local controller is envisaged to only get updates from the main global controller if it is coming alive due to having been down or when taking over another down node. If a node goes down, the sink node will connect to the next available controller node. To achieve the updated state, the new controller will execute an Anti-entropy synchronisation with the global controller; however, this would be rare, given the nature of updates in SDWSN. The sensory data is mostly upstream and therefore most updates will be from the local controller to the global controller. This algorithm improves the time total time complexity from $O(n \log n)$ to $O(n)$. The anti-entropy

protocol between the local controller and the global controller is depicted in Algorithm 5 and in Figure 4.4 the flow chart detailing the steps of the algorithm is shown. The flowchart reflects the steps that the algorithm undertakes during execution.

Algorithm 5 Anti-entropy algorithm with Fragmentation [8]

Require: Let S be a set of participants: $S = \{p, q, r, \dots\}$

```

for every  $T$  period do
    Synchronise with the global controller: send state  $O(n)$ 
    Send state:  $(v, t_i$  to  $q$  with state  $(v, t)$  Update
end for

Upon receiving  $p.state$   $O(1)$ 
if  $p.state.time > q.state.time$  i.e  $t$  then
     $q.state : (v, t) \leftarrow p.state : (v, t_i)$ 
else if  $p.state.time : (v, t_i) < q.state.time : (v, t)$  then
    send a reply to  $p$ 
end if

Upon receiving a reply  $O(1)$ 
if  $p.state.time < q.state.time$  then
     $p.state : (v, t_i) \leftarrow q.state.time : (v, t)$ 
end if

if  $p.stae : (v, t) \in S$  such that  $(v_i, t_j) \notin q$  then  $O(1)$ 

    if sender is global controller then
        ignore
    else
        request  $(v_i, t_j)$ 
    end if
end if

Total time complexity  $O(n) + O(1) + O(1) + O(1)$   $O(n)$ 

```

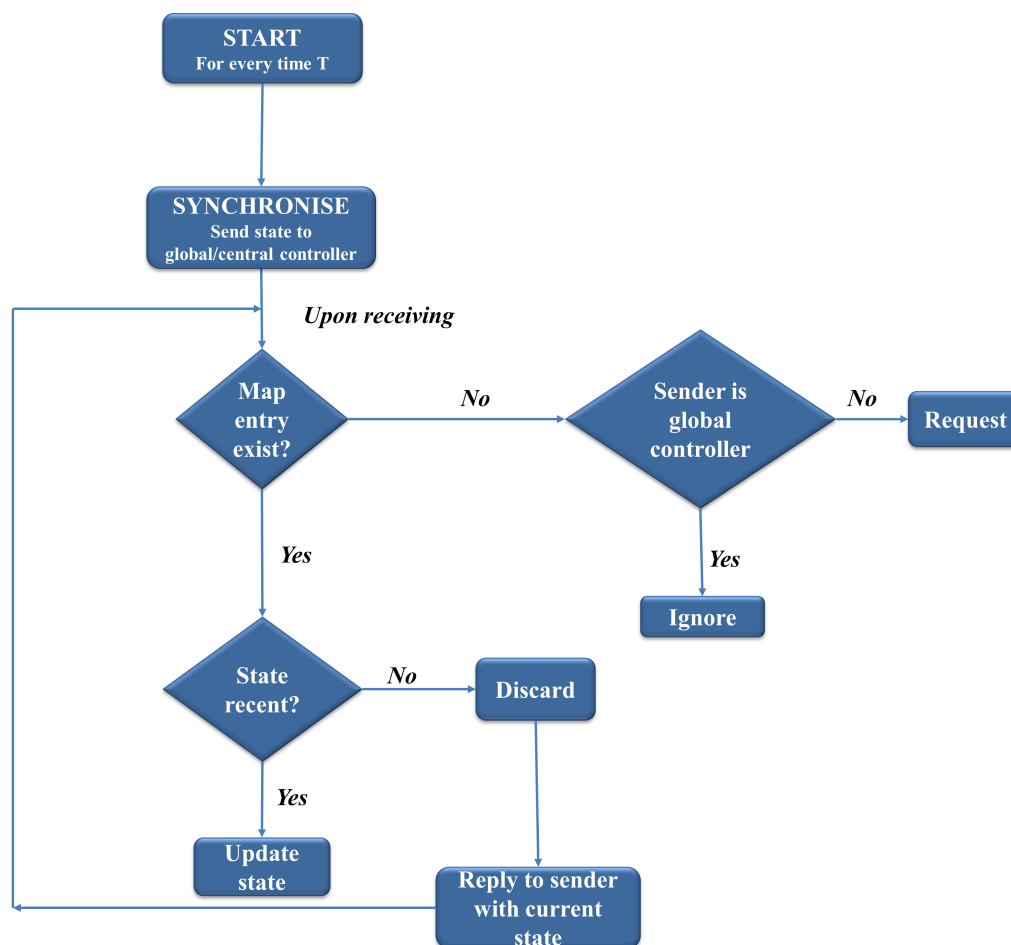


Figure 4.4. The flowchart of the Anti-entropy algorithm with fragmentation [8].

As stated above, there are similarities between these two algorithms. The main difference is in the initiation stages, the Best Effort algorithm is triggered by an event while the Anti-entropy algorithm is periodic. The similarities are in the information exchange between the nodes. As shown in Fig. 3 and Fig. 4, the contents of the algorithms are similar.

4.2.2 Time complexity

Time complexity is the amount of time or steps an algorithm takes to run as a function given the length of the input. The more complex an algorithm is, the longer it takes to run. Although many scholars consider it insignificant or negligible because of the advanced processing capabilities of modern systems, it is nevertheless vital to the SDWSN as it services time-sensitive applications. The

complexities of the above are listed alongside the algorithms. The Best Effort algorithm which sends updates to all peers upon a new event runs at $O(n)$; after the change which sends all updates to the global controller; the time complexity reduces to $O(1)$. On the other hand the anti-entropy algorithm changes from $O(n \log n)$ to $O(n)$. Thus, the proposed system satisfies some of the gossip algorithm/protocol characteristics and not all as identified by Montresor and listed above. There is a major shift with the proposed system which minimises the complexity of the peer-sampling service. The periodic pairing of and exchange between nodes are now between the local controller and the global controller. This reduces the complexities of the random peer selection.

4.3 CONCLUSION

The distributed-control system is very important for the efficacy of SDWSN networks. The choice of the method of distribution, consistency data model, and algorithms depends on the type of network and data represented. This chapter proposes a distributed-control system using fragmentation. Fragmentation entails a two-level control structure consisting of local controllers closer to the infrastructure elements, as well as a global controller overseeing the whole network. To realise fragmentation, the eventual consistency model, which uses Best Effort and Anti-entropy algorithms is adopted. The two algorithms are adapted to achieve fragmentation. Time complexity is of the original model and the fragmentation is computed and accordingly, the fragmentation model proves efficient. Whilst time complexity provides theoretical proof of efficiency, this still needs to be validated and qualified practically. The next chapter deals with the implementation and evaluation details of the proposed system.

CHAPTER 5 RESULTS AND DISCUSSION

In Chapter 4, we proposed and presented a distributed-control system for the software-defined wireless sensor networks using fragmentation. Hypothetically, the two algorithms used in the formation of the fragmentation model have an efficient time complexity as shown in the big O-notation. The big O notation is a computer science model used to measure the performance or complexity of an algorithm. This chapter evaluates the veracity of the hypotheses with practical implementation of the proposed model to verify the viability of the fragmentation for an SDWSN control system. In this chapter, we also present and discuss the results obtained from the evaluation. The rest of the chapter is organised as follows: Section 5.1 highlights the methodology used to carry out the experiments; we also discuss the tools used, the experimental setup, the evaluation methods and procedures, and the simulation setup respectively. Section 5.2 present the results obtained from the experiments and provide a detailed discussion of the results. The results are categorised in different metrics used in the evaluation, which are the controller setup time, throughput of the packets, the latency through the round trip time, the standard deviation, the packet error rate, and the time variations. We discuss the challenges encountered during the experimental evaluations in Section 5.3.

5.1 EXPERIMENTAL EVALUATION

5.1.1 Tools

The proposed model is implemented on the control plane of the SDWSN stack. It requires an SDN-based sensor node as well as an SDWSN controller. The SDN-WISE solution, which consists of an ONOS controller and a Cooja simulator is used for this purpose. The ONOS framework is traditionally an enterprise SDN solution, adapted to SDWSN by Gallucio *et al.* [136]. A Cooja adaptation was

also made to ensure the simulation of the SDN-based WSN. The fragmentation model is implemented inside the ONOS controller.

ONOS is a cluster-based distributed SDN controller, which can also operate in singular mode. The adaptation to SDWSN was tested in this mode; the focus though was on heterogeneity between a WSN network and an Ethernet network. Kobo *et al.* [189] tested this solution by using multiple controllers in a distribution setup. SDN-WISE 1 is implemented on ONOS version 1.0.2 and therefore, our proposed solution is also based on these versions. There is a mastership service in ONOS where the active controller becomes a master and the other controllers in the cluster become slaves. In the event of failure, the remaining slaves elect a master. This service is kept as is in the proposed system, thus each local controller is the master while the others become slaves.

In ONOS, clusters are formed out of one or more ONOS instances. Each ONOS instance is a controller. When the ONOS instance operates as a single controller; it uses the ONOS-trivial module. When more than one instances are used in a clustered form, ONOS-core is used instead. ONOS is modular and based on the OSGi Java framework. The ONOS subsystem is built using OSGi's Apache Karaf container. To create a cluster, ONOS uses two scripts in `onos-form-cluster` and/or uses test cells. The `onos-form-cluster` script merges two or more ONOS instances into a cluster. A test cell is a controller cluster environment for testing purposes. ONOS uses Hazelcast [207] to manage the distributed nodes in a cluster. Hazelcast is an in-memory data grid that is used to manage distributed data stores.

5.1.2 Experimental Setup

Figure 5.1 depicts the experiments that were carried out in evaluating the viability of the fragmentation model for SDWSN control. Three test experiments namely, experiments *A*, *B*, and *C*, were conducted comparatively; a single central controller, a distributed controller, and a distributed controller with fragmentation respectively. Figure 5.1(a) depicts a single central controller, referred herein as central. This experiment consisted of the ONOS instance operating in single mode and is labelled experiment *A*. The second experiment *B*, depicted in Figure 5.1(b), shows a distributed controller running ONOS instances in a clustered format, herein referred to as original ONOS. In Figure 5.1(c), the fragmentation model is represented in a distributed-control structure. This is the implementation of the proposed model in ONOS. The experiments are varied for the purposes of evaluating impact and efficiency. First,

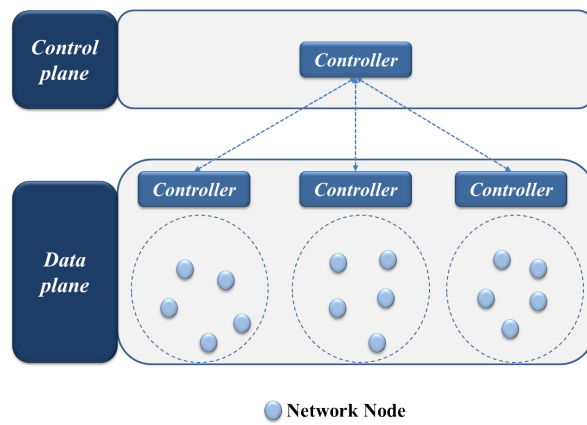
we evaluate the impact of a distributed-control system in SDWSN by comparing experiment *A* and *B*. Although this determination was established in [189] and [8], this seeks to extend the evaluation by increasing the sampling time and varied experiments. Secondly, the two distributed-control systems, *B* and *C*, seek to determine the efficiency of the fragmentation model. Therefore the comparative experimentation seeks to answer two questions first, if a distributed control system is necessary and secondly, if fragmentation indeed brings about the efficiency sought.

All the experiments were run in independent virtual machines (VM). The SDWSN simulations running the Cooja SDN emulated nodes were conducted from a VM with 2GHz CPU and 2G RAM. In experiment *A*, the controller was run in a VM consisting of 2GHz CPU and 2G RAM. The three VMs used in experiment *B* for the controllers consisted of 2GHz CPU and 1G RAM. In experiment *C*, three controllers were used in a clustered mode operating as local controllers, as well as one global controller. The three controllers were run in different VMs, all with the specification of 2GHz CPU and 1G RAM; the global controller VM had 2GHz CPU and 2G RAM.

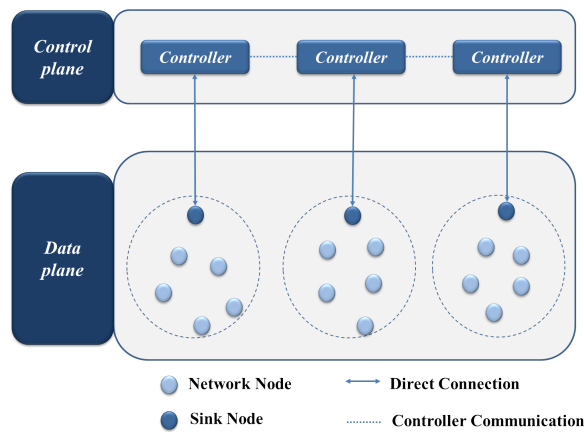
5.1.3 Evaluation Methods and Procedures

The SDWSN simulation consists of emulated SDN-enabled sensor nodes and sink nodes. The sensor nodes communicate by using the IEEE 802.15.4 [2] wireless communication specification. The sink nodes communicate with the other sensor nodes by using the same IEEE 802.15.4 medium and communicate with the controller(s) by using the Contiki IP stack. The sink nodes act as gateways between the sensor nodes and the controllers. The IP stack is based on IPv6's 6LowPAN [134]. The controllers use IPv4 interface; there is internal IP tunnelling that enables the sink nodes to communicate with the controllers. The sink nodes assume the IP address of the host VM.

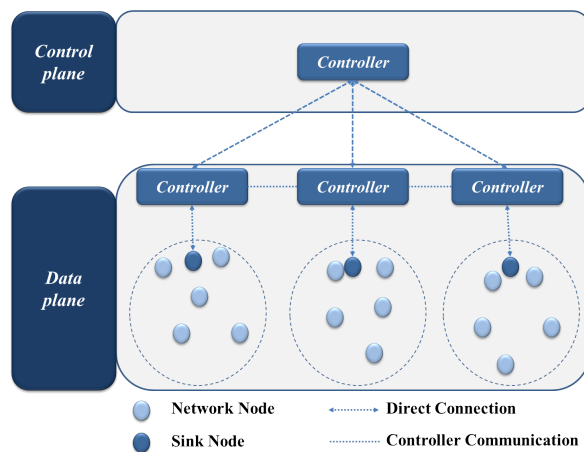
Three sink nodes are used across all experiments. The sink-to-controller mapping is initialised at 1:1 except in the single controller experiment where it is 3:1. The configuration of the sensor nodes varies between sample sizes of 24, 30, and 39 nodes. This is referred to as scenarios, thus scenario 24, scenario 30, and scenario 39. The composition of the scenarios consisted of the three sink nodes, each having an equal share of the sensor nodes. Thus, in scenario 24, there were eight (8) sensor nodes for each sink node, and scenario 30 consisted of ten (10) sensor nodes for each sink node, whilst in scenario 39, thirteen (13) sensor nodes were connected to each sink node. These three scenarios were



(a) Experiment A, single central controller.



(b) Experiment B, distributed controllers with ONOS original.



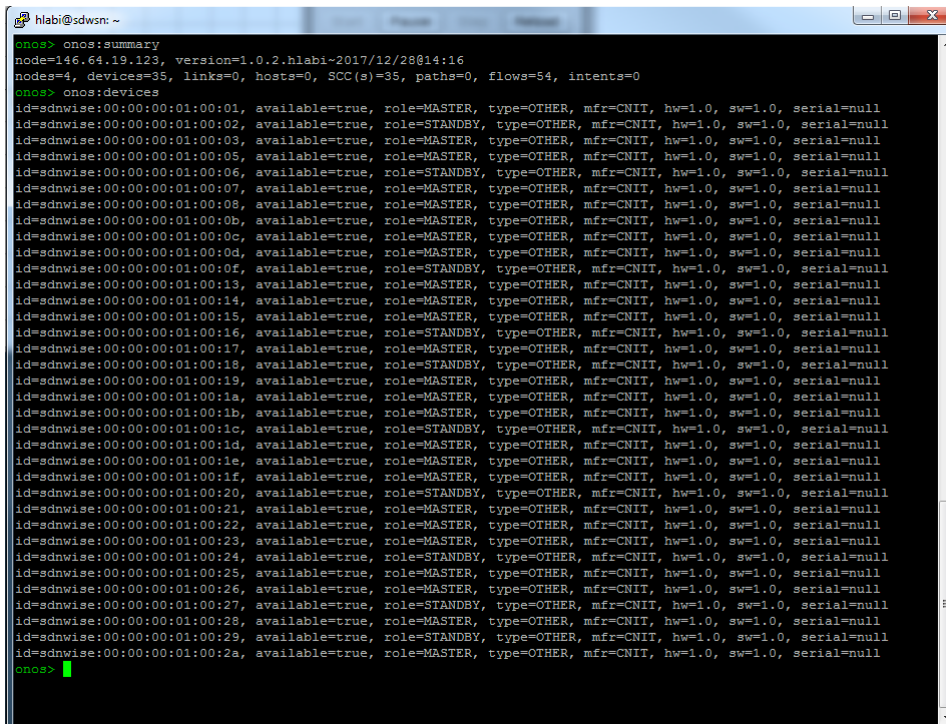
(c) Experiment C, distributed controller with fragmentation.

Figure 5.1. The experimental design and setup.

run for 30 minutes in each experiment. Thereafter, we ran scenario 39 again for 1 hour and 5 hours, these are referred to as scenario 1 hour and scenario 5 hours. The experiments were conducted during a holiday when the network traffic was at its lowest.

5.1.4 Simulation

The ONOS controller has been customised for SDWSN by Galluccio *et al.* [136] for the SDN-WISE solution. Here, it is used to control the SDWSN simulation. The following figures show the ONOS controller during the simulation tests. An illustration is made with one controller. Figure 5.2 shows the identity of the controller with some of the connected devices, while Figure 5.3 shows the flows during testing.



```

hlabi@sdwsn: ~
onos> onos:summary
node=146.64.19.123, version=1.0.2.hlabi-2017/12/28@14:16
nodes=4, devices=35, links=0, hosts=0, SCC(s)=35, paths=0, flows=54, intents=0
onos> onos:devices
id=sdnwise:00:00:00:01:00:01, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:02, available=true, role=STANDBY, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:03, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:05, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:06, available=true, role=STANDBY, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:07, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:08, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:0b, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:0c, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:0d, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:0f, available=true, role=STANDBY, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:13, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:14, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:15, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:16, available=true, role=STANDBY, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:17, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:18, available=true, role=STANDBY, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:19, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:1a, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:1b, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:1c, available=true, role=STANDBY, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:1d, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:1e, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:1f, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:20, available=true, role=STANDBY, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:21, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:22, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:23, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:24, available=true, role=STANDBY, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:25, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:26, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:27, available=true, role=STANDBY, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:28, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:29, available=true, role=STANDBY, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
id=sdnwise:00:00:00:01:00:2a, available=true, role=MASTER, type=OTHER, mfr=CNIT, hw=1.0, sw=1.0, serial=null
onos>

```

Figure 5.2. The identity of the controller with connected devices.

```

hlabi@sdwsn: ~
onos> onos:flows
deviceId=sdnwise:00:00:01:00:01, flowRuleCount=3
id=100004cf5c2ac, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.net.proxyarp
selector=[ETH_TYPE{ethType=806}]
treatment=[OUTPUT(port=CONTROLLER)]
id=200004ce65aa6, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.provider.lldp
selector=[ETH_TYPE{ethType=88cc}]
treatment=[OUTPUT(port=CONTROLLER)]
id=200004ce668f0, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.provider.lldp
selector=[ETH_TYPE{ethType=8942}]
treatment=[OUTPUT(port=CONTROLLER)]
deviceId=sdnwise:00:00:01:00:02, flowRuleCount=3
id=10000f92eb798, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.net.proxyarp
selector=[ETH_TYPE{ethType=806}]
treatment=[OUTPUT(port=CONTROLLER)]
id=200004ce65e67, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.provider.lldp
selector=[ETH_TYPE{ethType=88cc}]
treatment=[OUTPUT(port=CONTROLLER)]
id=200004ce66cb1, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.provider.lldp
selector=[ETH_TYPE{ethType=8942}]
treatment=[OUTPUT(port=CONTROLLER)]
deviceId=sdnwise:00:00:01:00:03, flowRuleCount=0
deviceId=sdnwise:00:00:01:00:05, flowRuleCount=0
deviceId=sdnwise:00:00:01:00:06, flowRuleCount=3
id=200004ce66d6b, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.provider.lldp
selector=[ETH_TYPE{ethType=88cc}]
treatment=[OUTPUT(port=CONTROLLER)]
id=200004ce67bb5, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.provider.lldp
selector=[ETH_TYPE{ethType=8942}]
treatment=[OUTPUT(port=CONTROLLER)]
id=300004cf5d571, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.provider.host
selector=[ETH_TYPE{ethType=806}]
treatment=[OUTPUT(port=CONTROLLER)]
deviceId=sdnwise:00:00:01:00:07, flowRuleCount=0
deviceId=sdnwise:00:00:01:00:08, flowRuleCount=3
id=100004cf5dcf3, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.net.proxyarp
selector=[ETH_TYPE{ethType=806}]
treatment=[OUTPUT(port=CONTROLLER)]
id=20000f91f6618, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.provider.lldp
selector=[ETH_TYPE{ethType=88cc}]
treatment=[OUTPUT(port=CONTROLLER)]
id=20000f91f7462, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.provider.lldp
selector=[ETH_TYPE{ethType=8942}]
treatment=[OUTPUT(port=CONTROLLER)]
deviceId=sdnwise:00:00:01:00:0b, flowRuleCount=0
deviceId=sdnwise:00:00:01:00:0c, flowRuleCount=0
deviceId=sdnwise:00:00:01:00:0d, flowRuleCount=0
deviceId=sdnwise:00:00:01:00:0f, flowRuleCount=3
id=10000f92f7acc, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.net.proxyarp
selector=[ETH_TYPE{ethType=806}]
treatment=[OUTPUT(port=CONTROLLER)]
id=20000ceb0ac73, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.provider.lldp
selector=[ETH_TYPE{ethType=88cc}]
treatment=[OUTPUT(port=CONTROLLER)]
id=20000ceb0babb, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.provider.lldp
selector=[ETH_TYPE{ethType=8942}]
treatment=[OUTPUT(port=CONTROLLER)]
deviceId=sdnwise:00:00:01:00:13, flowRuleCount=0
deviceId=sdnwise:00:00:01:00:14, flowRuleCount=3
id=100004cf6424e, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.net.proxyarp
selector=[ETH_TYPE{ethType=806}]
treatment=[OUTPUT(port=CONTROLLER)]
id=200004ce6da48, state=PENDING_ADD, bytes=0, packets=0, duration=0, priority=40000, appId=org.onosproject.provider.lldp

```

Figure 5.3. The flows inside the controller during evaluation.

The SDWSN simulation uses the Cooja adaptation as in the SDN-WISE solution. The illustration is made with one experiment which is experiment C, scenario 30; any scenario could have been used for illustration. Figure 5.4 shows the node structure of the simulation, the green motes are the sink nodes while the orange motes are the sensor nodes. The middle circle shows the reach of the cluster, this is to ensure parallelism amongst the clusters.

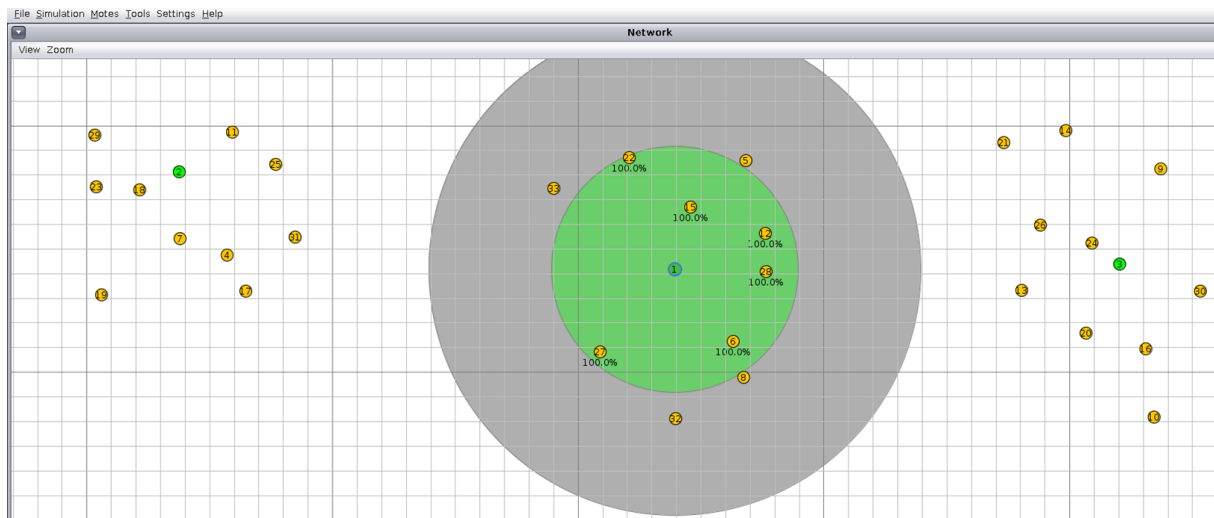


Figure 5.4. The node structure under test.

Figure 5.5 shows the nodes the simulation script used to time the experiment, in this case, a period of 30 minutes was used which equated to 1800000 seconds.

```

1  /*
2  * Example Contiki test script (JavaScript).
3  * A Contiki test script acts on mote output, such as via printf()'s
4  * The script may operate on the following variables:
5  * Mote mote, int id, String msg
6  */
7
8  TIMEOUT(1800000);
9
10 while (true) {
11   log.log(time + ":" + id + ":" + msg + "\n");
12   YIELD();
13 }

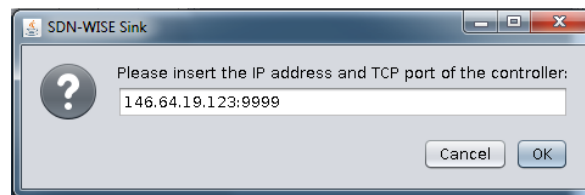
```

Figure 5.5. The simulation-timing script.

Figure 5.6(a) shows the simulation control window, which enabled us to set the speed of the simulation, as well as to start, pause, and reload. Once started, the prompt in Figure 5.6(b) appeared which required the IP address of the controller. This is the manual operation alluded to in the coming sections.



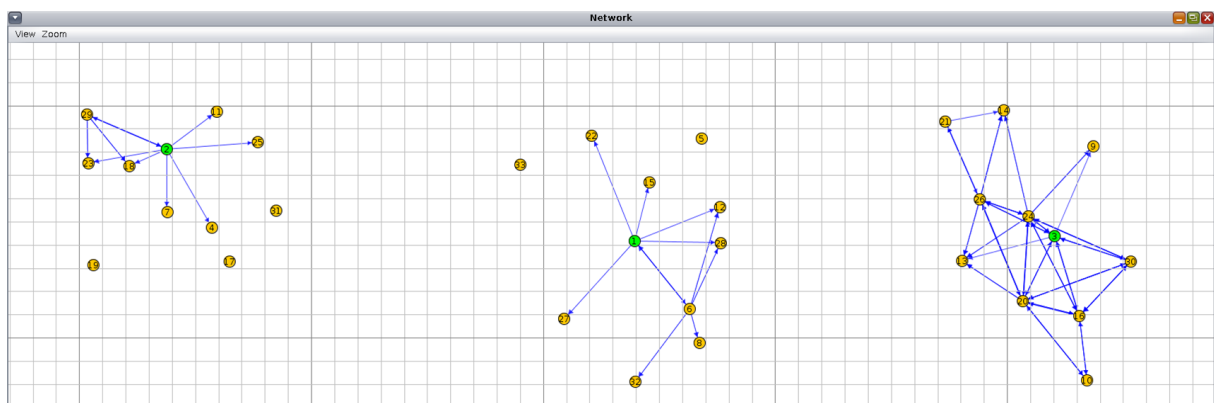
(a) Simulation control window.



(b) Sink connection prompt window.

Figure 5.6. The simulation control windows.

Figure 5.7 depicts the node structure in operation, when data is being exchanged amongst the nodes, sensor and sink nodes; this depicts the IEEE 802.15.4 physical communication referred to in Chapter 2 Subsection 2.5.1.1. The sink nodes then convey the data to the controller, creating flows.

**Figure 5.7.** The exchange of data amongst the nodes.

In Figure 5.8 and Figure 5.9, the snapshots of the radio messages and duty cycles of the nodes during testing are shown respectively. The nodes show a 100% ON with the percentages of transmit (Tx) and

receive (Rx); the average is at the bottom. Lastly, Figure 5.10 shows the full picture of the simulation during testing. The window at the bottom shows the timeline of the nodes.

No.	Time	From	To	Data
1	05:29.407	31	[5 d]	12: 0x010CFFFF 001F0163 0002038D
2	05:29.409	31	[5 d]	31: 0x011F0002 001F0263 000403...
3	05:29.410	4	[7 d]	31: 0x011F0002 001F0262 000703...
4	05:29.411	7	[9 d]	31: 0x011F0002 001F0261 000203...
5	05:29.473	5	[4 d]	12: 0x010CFFFF 00050163 0001028D
6	05:30.373	17	[3 d]	12: 0x010CFFFF 00110163 0002028C
7	05:30.419	28	[6 d]	12: 0x010CFFFF 001C0163 0001018C
8	05:30.421	33	[6 d]	12: 0x010CFFFF 00210163 0001038D
9	05:30.422	28	[6 d]	31: 0x011F0001 001C0263 000101...
10	05:30.424	33	[4 d]	19: 0x01130001 00210263 001603...
11	05:30.425	22	[4 d]	19: 0x01130001 00210262 000F03...
12	05:30.427	15	[5 d]	19: 0x01130001 00210261 000103...
13	05:30.438	32	[3 d]	12: 0x010CFFFF 00200163 0001038D
14	05:30.440	32	[3 d]	16: 0x01100001 00200263 000803...
15	05:30.441	8	[3 d]	16: 0x01100001 00200262 000603...
16	05:30.443	6	[5 d]	16: 0x01100001 00200261 000103...
17	05:30.453	12	[5 d]	12: 0x010CFFFF 000C0163 0001028C
18	05:30.462	25	[5 d]	12: 0x010CFFFF 00190163 0002028C
19	05:30.465	13	[4 d]	12: 0x010CFFFF 000D0163 0003028C
20	05:30.466	23	[5 d]	12: 0x010CFFFF 00170163 0002018C
21	05:30.468	23	[5 d]	31: 0x011F0002 00170263 000201...

Figure 5.8. The radio messages of the motes.

Mote	Radio on (%)	Radio TX (%)	Radio RX (%)
SDN-WISE Mote 1	100.00%	0.01%	0.15%
SDN-WISE Mote 2	100.00%	0.01%	0.13%
SDN-WISE Mote 3	100.00%	0.02%	0.14%
SDN-WISE Mote 4	100.00%	0.01%	0.10%
SDN-WISE Mote 5	100.00%	0.03%	0.09%
SDN-WISE Mote 6	100.00%	0.03%	0.14%
SDN-WISE Mote 7	100.00%	0.02%	0.07%
SDN-WISE Mote 8	100.00%	0.00%	0.05%
SDN-WISE Mote 9	100.00%	0.02%	0.04%
SDN-WISE Mote 10	100.00%	0.02%	0.12%
SDN-WISE Mote 11	100.00%	0.02%	0.11%
SDN-WISE Mote 12	100.00%	0.02%	0.09%
SDN-WISE Mote 13	100.00%	0.02%	0.08%
SDN-WISE Mote 14	100.00%	0.04%	0.09%
SDN-WISE Mote 15	100.00%	0.02%	0.09%
SDN-WISE Mote 16	100.00%	0.02%	0.07%
SDN-WISE Mote 17	100.00%	0.02%	0.13%
SDN-WISE Mote 18	100.00%	0.00%	0.07%
SDN-WISE Mote 19	100.00%	0.02%	0.14%
SDN-WISE Mote 20	100.00%	0.02%	0.08%
SDN-WISE Mote 21	100.00%	0.02%	0.08%
SDN-WISE Mote 22	100.00%	0.02%	0.15%
SDN-WISE Mote 23	100.00%	0.02%	0.11%
SDN-WISE Mote 24	100.00%	0.02%	0.08%
SDN-WISE Mote 25	100.00%	0.04%	0.11%
SDN-WISE Mote 26	100.00%	0.00%	0.03%
SDN-WISE Mote 27	100.00%	0.02%	0.13%
SDN-WISE Mote 28	100.00%	0.02%	0.05%
SDN-WISE Mote 29	100.00%	0.02%	0.07%
SDN-WISE Mote 30	100.00%	0.01%	0.11%
SDN-WISE Mote 31	100.00%	0.02%	0.05%
SDN-WISE Mote 32	100.00%	0.02%	0.02%
SDN-WISE Mote 33	100.00%	0.02%	0.09%
AVERAGE	100.00%	0.02%	0.09%

Figure 5.9. The duty cycles of the motes.

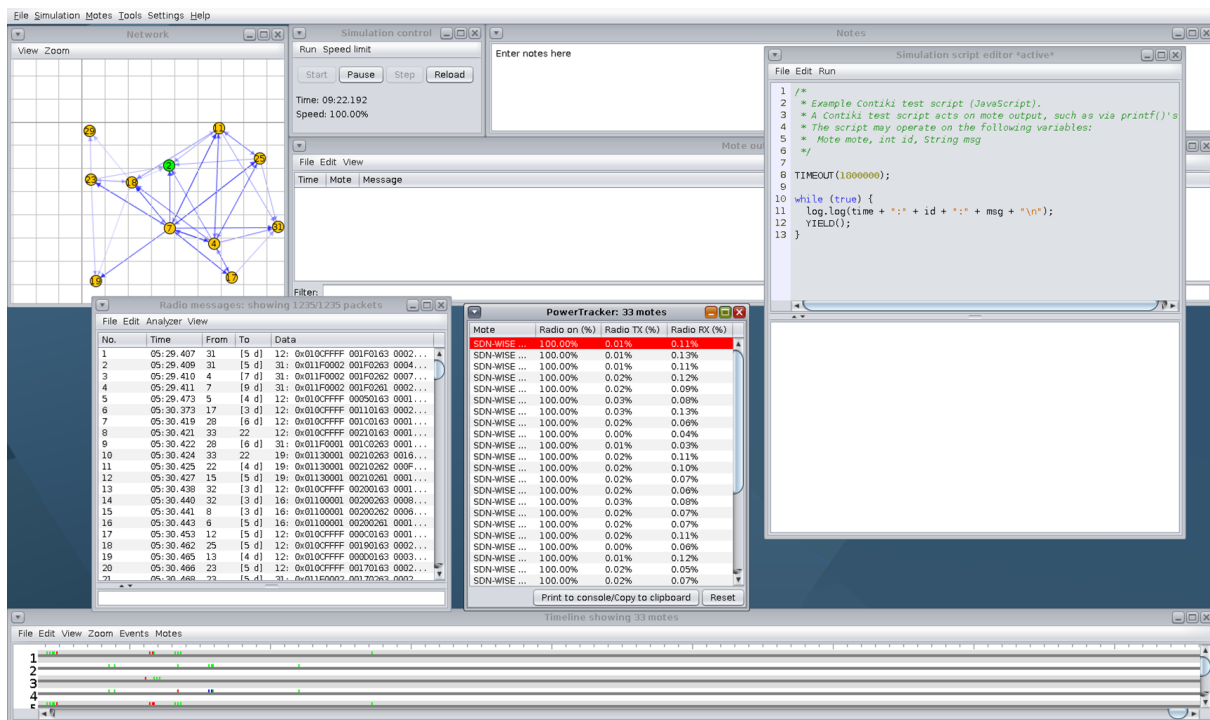


Figure 5.10. The test simulation.

5.2 RESULTS AND DISCUSSION

This experimental evaluation considered three main metrics for this research work in Round-Trip Time (RTT) or Round-Trip Delay, Standard Deviation and Packet Error rate. The RTT measures the time it takes for a packet to traverse from the simulation to the controller and back. The standard deviation measures the variation of the time (RTT) or the delay across all packets recorded over the evaluation period. This assists in determining the relational proportion of consistency or lack of it in the performance. The packet error rate looks at the rate of packet loss. All the results are presented and recorded in Table 5.1 and further presented in the following sections.

5.2.1 Controller setup time

The controller setup time is measured by the time it takes for the sink node to negotiate a connection to the controller. This is done by observing the SYNC packet, which is the first packet to be sent to the

Table 5.1. The results of the experiments

Experiments	A			B			C			B-extended		C-extended	
	24	30	39	24	30	39	24	30	39	1hour	5hour	1hour	5hour
RTT average	1043.50	990.88	960.05	673.61	530.25	551.60	629.67	525.30	538.27	553.80	468.84	492.46	495.81
RTT min	220.84	224.63	209.73	181.22	167.48	170.28	170.37	192.85	212.34	170.93	158.74	141.15	152.71
RTT max	12410.79	16256.76	15344.11	26545.49	25892.78	11816.93	5989.72	4231.52	6911.58	9920.48	8957.08	4989.73	2382.29
RTT median	695.62	608.53	572.97	500.56	410.57	451.11	528.78	467.16	456.70	433.42	427.37	441.91	452.12
RTT std dev	1089.80	1157.05	1301.06	1118.12	975.15	550.36	536.71	322.69	433.53	619.12	273.56	304.28	218.33
Packet error	7.00	18.00	13.00	11.00	20.00	17.00	5.00	6.00	12.00	18.00	23.00	14.00	7.00
PE rate(%)	0.37	0.84	0.43	0.59	0.90	0.56	0.26	0.26	0.39	0.51	0.29	0.39	0.09
SYNC1	1291.58	836.50	1091.93	469.97	522.59	365.54	572.72	514.38	232.42	475.06	323.79	484.27	583.71
SYNC2	1095.73	566.16	1310.53	448.34	489.54	495.01	398.57	531.54	402.62	491.21	530.97	661.47	391.08
SYNC3	905.70	1170.94	507.32	337.28	405.86	395.22	385.68	425.06	466.21	604.99	489.60	492.90	733.92
SYNC average	1097.67	857.87	969.93	418.53	472.66	418.59	452.32	490.33	367.08	523.75	448.12	546.21	569.57
Packets	1893.00	2135.00	2991.00	1875.00	2223.00	3025.00	1895.00	2285.00	3061.00	3547.00	7909.00	3555	7927.00

controller by the sink node. The experiments consisted of three sink nodes which all had to make a connection to the controller, one after the other.

The first experiment exhibited high connection time in all scenarios. The average times of the three sink nodes were 1097.67, 857.87, and 969.93 nanoseconds for scenarios 24, 30, and 39 respectively for experiment *A*. In experiment *B*, the average synchronisation times were 418.53, 472.66, and 452.32 nanoseconds for scenarios 24, 30, and 39 respectively. Experiment *C* took 452.32, 490.33, and 367.08 nanoseconds average times across the respective scenarios. The distributed controller experiments took less time to connect than the central controller experiment. Figure 5.11 depicts the graphical representation of these results.

The controller setup time is critical as it determines the time and delay dynamics that could affect the operation of the network. This measure also indicates the controller response time which is a very important metric to determine the efficiency of the controller. In startup, this helps to determine the time it would take for a network to start operating. In a case of undesired events such as controller failure, it determines the time to make a reconnection to another controller. Other scenarios include new nodes joining the network. As expected, the central controller took more time than the distributed experiments to make connections. This could be worse if the number of nodes and sink nodes increased in the network. Therefore, a central controller is not ideal. The distributed controllers improve this time by almost 50%. This is mainly because the distributed controllers share the load. Furthermore, the fragmentation also introduces independent processing amongst the clustered controllers. The results

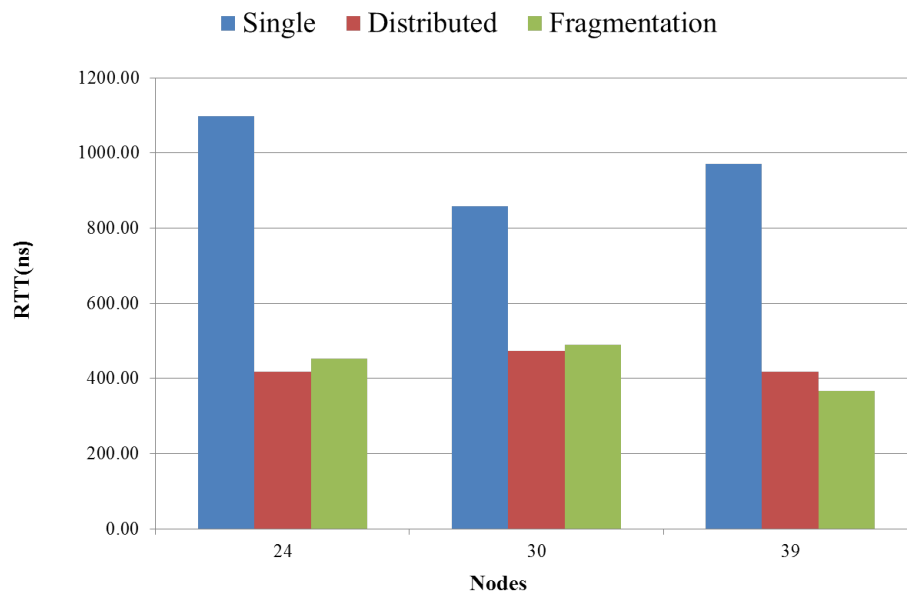


Figure 5.11. Controller setup times of experiments *A*, *B*, and *C*.

show that the distributed experiments are on par in terms of the controller setup time. However, there is a minor difference which cannot be qualified and can largely be attributed to normal networking dynamics. However, the independent processing fostered by the fragmentation model will bring about efficiency in a large network where the process is automated. Currently, the sink nodes are connected manually to the controllers. This is a challenge that should be investigated further to ensure that the simulation allows more nodes and connects to the controllers systematically.

5.2.2 Number of packets

The number of packets delivered between the controller(s) and the sink nodes differs for various reasons such as the number of sensor nodes, the duration of the experiment, and lastly the architecture. The single central controller exhibited the smallest number packets, followed by the distributed ONOS while the fragmentation distribution was better. The number of packets recorded for experiment *A* are 1893, 2135, and 2991 for scenarios 24, 30, and 39 respectively. The three scenarios in experiment *B* produced 1875, 2223, and 3025 respectively. Experiment *C* had more, 1895, 2285, and 3061 packets across the respective scenarios. These results are summarised in Table 5.1 and depicted in Figure 5.12.

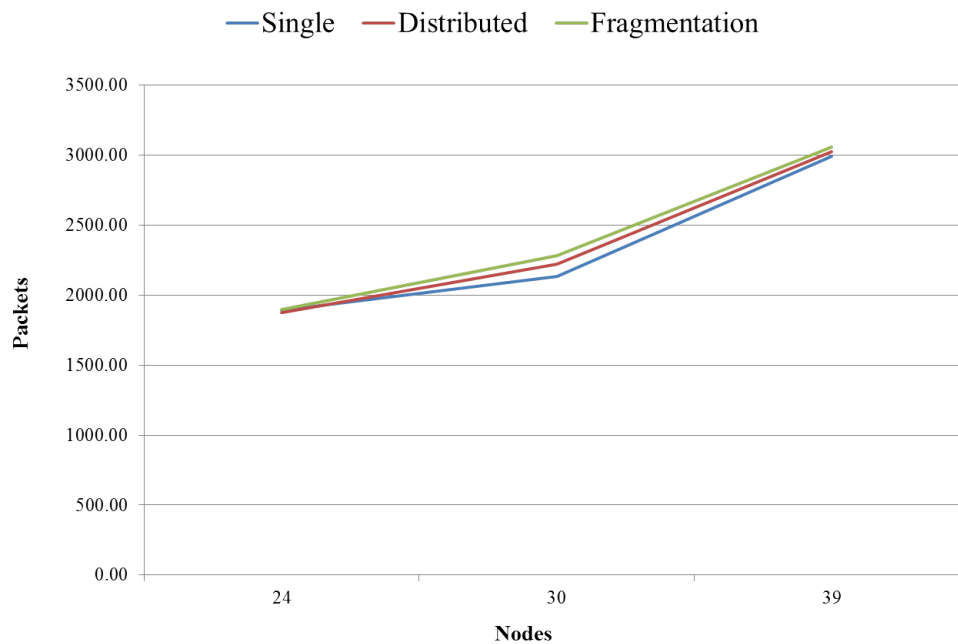


Figure 5.12. Number of packets produced.

The different scenarios produced different numbers of packets. The number of packets is determined by the number of nodes used in simulation, the duration of the test, and the efficiency of the controller. The first and the second are apparent but the third requires validation. As the number of nodes and the duration of the test increase, the number of packets increases as well. However, the controller also plays a huge role in this determination because the time it takes to process the packet makes a difference. Thus, an efficient controller results in efficient throughput of packets. The simulation produces packets at a constant rate and speed but the throughput of the packets to and from the controller has a vital impact on this rate. The central controller setting in experiment A produced the lowest number of packets compared to the distributed experiments. This is mainly because the load is shared amongst the clustered controllers in the distributed setting. The fragmentation model produced more packets than the other distributed model in the experiment. This is largely attributed to the efficiency of the algorithms as shown in Chapter 4. This validates the big O notation which showed that the algorithms used in fragmentation have a better time complexity. This improvement is also apparent in the extended experiments. The reduced distance between the controller and the sink node reduces the pressure from the low data rate capacity of the sink node, thereby enabling improved throughput.

5.2.3 Round trip time (RTT)

The RTT measures the time or delay a packet takes to make a round trip from a sink node to the controller and back. The experiments showed a higher RTT on the central controller experiment than on the distributed experiments. The average RTTs in experiment *A* for scenarios 24, 30, and 39 were 1043.50, 990.88, and 960.05 nanoseconds respectively. At the first distributed setting, experiment *B*, the three scenarios, 24, 30 and 39 produced 673.61, 530.25, and 551.60 nanoseconds respectively. At the second distributed setting, the fragmentation model, the RTTs were slightly lower with 629.67, 525.30, and 538.27 nanoseconds for the respective scenarios. The minimum RTTs exhibited in experiment *A* were 220.84, 224.63, and 209.73 nanoseconds whilst experiment *B* exhibited 181.22, 167.48, and 167.48 nanoseconds across scenarios 24, 30, and 39 respectively. Experiment *C*, on the other hand, exhibited the lowest minimum RTTs of 170.37, 192.85, and 212.34 across the respective scenarios. The highest maximum RTTs were experienced in experiments *A* and *B*; with *A* having the maximum RTTs of 12410.79, 16256.76, and 15344.11 nanoseconds for the respective scenarios while *B* had 26545.49, 25892.78, and 11816.93 nanoseconds. Experiment *C* had the lowest maximum RTTs in 5989.72, 4231.52, and 6911.52 for scenario 24, 30, and 39 respectively. The median RTTs in experiment *A* were 695.62, 608.32, and 572.97 across all scenarios respectively. In experiment *B*, the median RTTs were 500.56, 410.57, and 451.11 nanoseconds, while experiment *C*'s medians were 528.78, 467.78, and 572.70 for all respective scenarios. Figures 5.13, 5.14, and 5.15 depict the graphical representation of the RTTs for scenarios 24, 30, and 39 respectively while 5.16 shows the graph of the average RTTs against the scenarios under testing.

The RTT is a very important measure to gauge the interaction between the SDWSN simulation and the controller. Although some research work, such as that of Erickson [139] and Dixit *et al.* [94] asserts that a controller can handle millions of packets per second, it is important to qualify this for SDWSN. The average RTT is high on the central controller as compared to that of the distributed versions. The central controller performs well in the initial stages and staggers as more packets come through. This can also be observed as the number of nodes increases. The results show an improvement in the distributed experiments. The results show an improvement by fragmentation, though the difference is not huge. The small difference in the RTT averages is due to the fact that the fragmentation model enabled more packets and thus the additional RTTs were added to the overall average. This is clearly highlighted in Figures 5.13, 5.14, and 5.15 which show that the fragmentation model is consistently below the other two models on the graphs. The other factors show this difference clearly such as

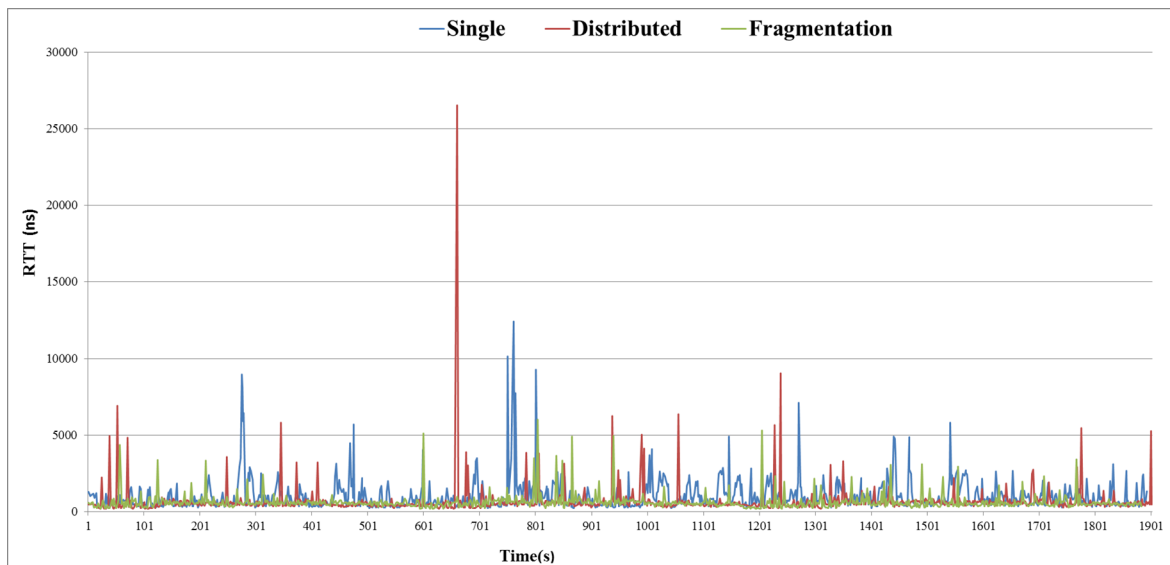


Figure 5.13. Average RTT for 24 nodes.

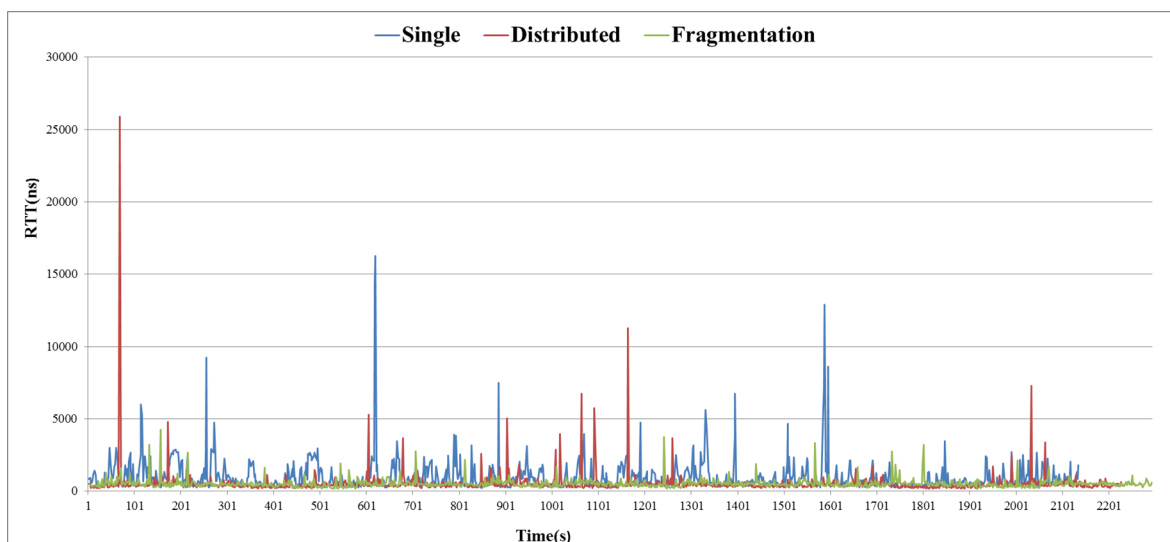


Figure 5.14. Average RTT for 30 nodes.

in the extended experiments which show fragmentation slightly below the original ONOS in terms of the average RTT. Although the overall RTT averages show a slight difference; the fragmentation model exhibits the best consistency as highlighted by the maximum and minimum values as depicted in Figures 5.13, 5.14, and 5.15.

The maximum RTT obtained on fragmentation is consistently relative as compared to the other experiments. On the other hand, the ONOS-distributed experiment and the central-controller experiment

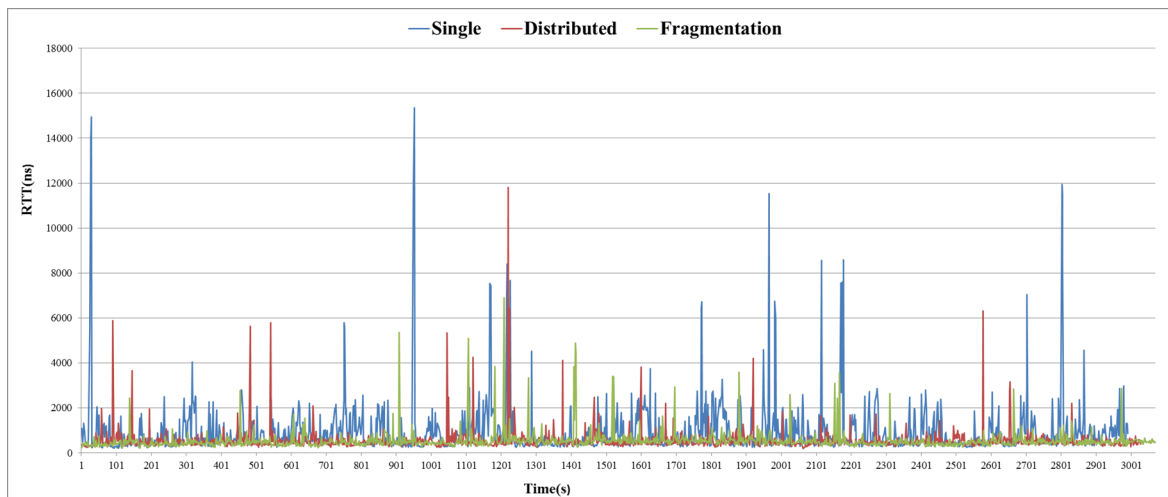


Figure 5.15. Average RTT for 39 nodes.

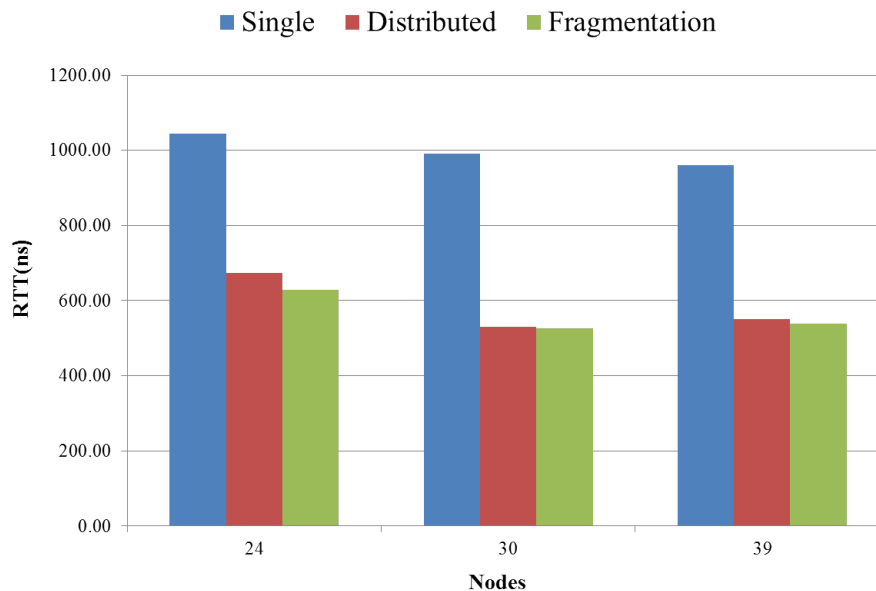


Figure 5.16. Average RTT vs number of nodes.

exhibited the highest peaks on the graphs. The minimum RTTs are, however, consistent across all the experiments. This is mainly because the first few packets go uninterrupted until the network gets busy. The median RTTs highlighted the middle value in all the RTTs observed, and as shown in Table 5.1, the distributed experiments showed a relatively consistent median lower than the central experiment. This shows that the disparity between the higher and the lower values is very low in distributed settings. Although limited in the scaling of the nodes, it can be concluded that scalability has a direct effect on RTT. The variation compared to the increase in nodes was minimal in the fragmentation experiment,

compared to the other two experiments. However, the real extent of the improvement can only be conclusively ascertained with an extended degree of scalability which at this stage could not be reached due to the limited capacity of the simulation tool.

5.2.4 Standard deviation

The standard deviation measures the variation of RTT over the testing period. The results obtained showed a low standard deviation for the distributed experiments. The central controller setting exhibited the highest standard deviation in 1089.80, 1157.05, and 1301.06 for scenarios 24, 30 and 39 respectively. Experiment *B*, which is the original ONOS in a distributed setting produced 1118.12, 975.15, and 550.36 for the respective scenarios while the fragmentation-based experiment *C* produced 536.71, 322.69, and 433.33 for scenarios 24, 30, and 39 respectively. Figure 5.17 depicts the graphical representation of the standard deviation.

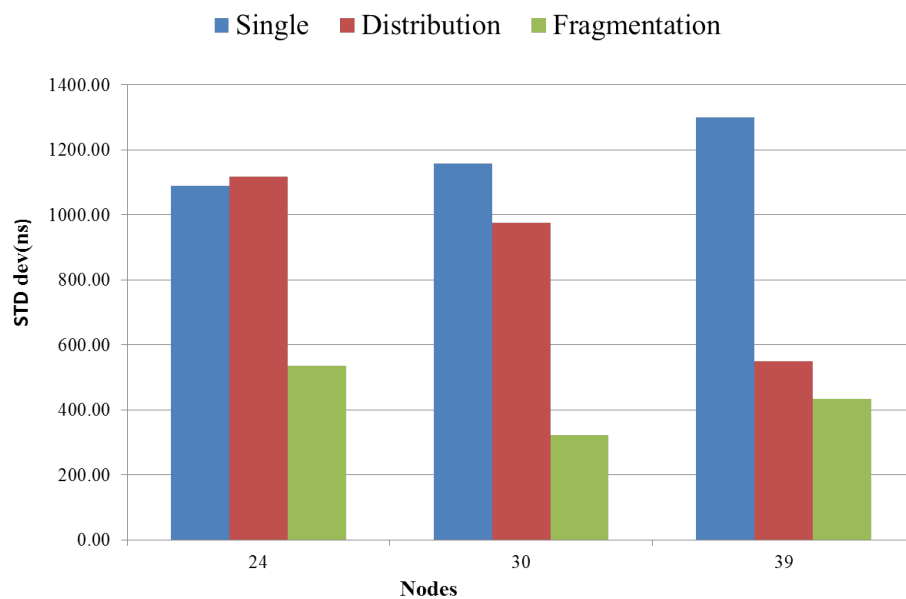


Figure 5.17. Average Standard deviation.

The variation trajectory in all the experiments is clearly observed in the standard deviation. The standard deviation is crucial as it highlights the degree of consistency in the performance of the various controller settings. The central controller exhibited the worst with an average of over 1000 ns, followed by the original ONOS, while the fragmentation exhibited very little variation. Greater

variation indicates inconsistent performance and, as shown by the results, the central controller fails in this regard. The two distributed experiments greatly reduce this variation which validates the idealism of a distributed-control system in the SDWSN control. The smallest variation observed was in the fragmentation model which validated its consistency in performance. This shows amongst other things, that the fragmentation model would be ideal for the dynamic SDWSN, as well as IoT.

The RTTs between the two distributed models do not differ much as the fragmentation model had more packets than the other distributed experiments and that added to the overall average. The standard deviation provides a clear indication because it calculates the variation of the RTTs and the quantity does not affect the ultimate value. Therefore, in addition to the difference in the number of packets sampled and averaged, the standard deviation provides a thorough account of the difference in performance. It shows that there is indeed an improvement in the fragmentation model. Another interesting aspect observed in the standard deviation as shown in Figure 5.17 is the fact that in experiment *A*, the central controller's standard deviation increases by the number of nodes while the ONOS-distributed experiment decreases the standard variation with the addition of nodes. The fragmentation model decreases the standard deviation too, but performs better in the 30-node scenario than the rest.

5.2.5 Packet error rate

The packet loss rate was minimally under 1% in all experiments and scenarios. Experiment *A* suffered 0.37, 0.84, and 0.43 packet loss for scenarios 24, 30, and 39 respectively. In experiment *B*, the losses for the respective scenarios were 0.59, 0.90, and 0.56. Packet loss rates of 0.26, 0.26, and 0.39 for scenarios 24, 30, and 39 were exhibited in experiment *C*. All these numbers are captured in Figure 5.18.

The overall packet error rate or loss was very low across all experiments and scenarios. The packet error rate was consistently under 1%, despite the variation in times and number of nodes. This can be attributed to the strong architectural maturity of the SDN-WISE model, as the distribution had little impact on it. The difference in the packet loss across the experiments was very minimal. The fragmentation model exhibited the lowest packet loss compared to the other two experiments. However, the packet loss was only experienced on the second sink node, the first and the third sink nodes had no loss at all. This was due to the fact that ONOS 1.0.2 is based on Hazelcast [207] to manage

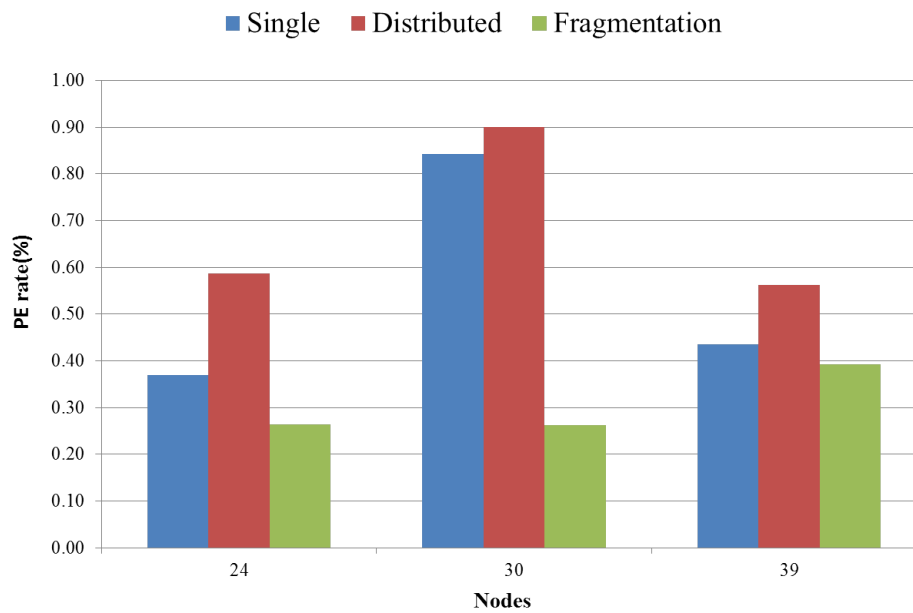


Figure 5.18. The Packet Error rate.

the distributed data stores. Hazelcast does not handle the split-brain situation well. Split-brain is a phenomenon originating from inconsistency between two or more different data sets (nodes), this is caused by either the design of the servers or synchronisation failures. Hazelcast historically only works well when the deployment of nodes is even, i.e. 2 or 4 nodes. The latest versions of ONOS have migrated to the Atomix [208] framework which uses RAFT [24], a consensus algorithm to handle split-brain situations and uses an odd number nodes, i.e. 3 or 5 nodes. Our experiments consisted of 3 nodes which was in direct contrast with the even allocation of nodes preferred by Hazelcast, hence the loss of packets from the second node. However, this warrants further and thorough investigation to confirm this because it also happened to the central node architecture. The single controller in ONOS does not use distributed functionalities. The increase in the number of sensor nodes did not have any technical impact on the packet error rate as the loss experienced is standard across all scenarios.

5.2.6 Time variations

The testing time of scenario 39 of the distributed-controller experiments, *B* and *C*, was extended from 30 minutes to one hour and five hours respectively. These two experiments are referred to as experiments *B*-extended and *C*-extended, as also shown in Table 5.1. These experiments only focused on the 39-node simulation because the purpose was to extend the testing period; the scaling of nodes

is represented by experiments *A-C*. The experiment was conducted to compare the two distributed experiments in an extended period. The controller setup time, average RTT, and minimum RTT are relative across the two experiments and the two scenarios, scenario 1 hour and scenario 5 hours. The results that vary between the two experiments are PE rate, maximum RTT and standard deviation. The maximum RTT for experiment *B-extended* showed 9920.48 and 8957.08 nanoseconds for scenarios 1 and 5 hour respectively, while experiment *C-extended* showed 4989.73 and 2382.29 nanoseconds for the same scenarios respectively.

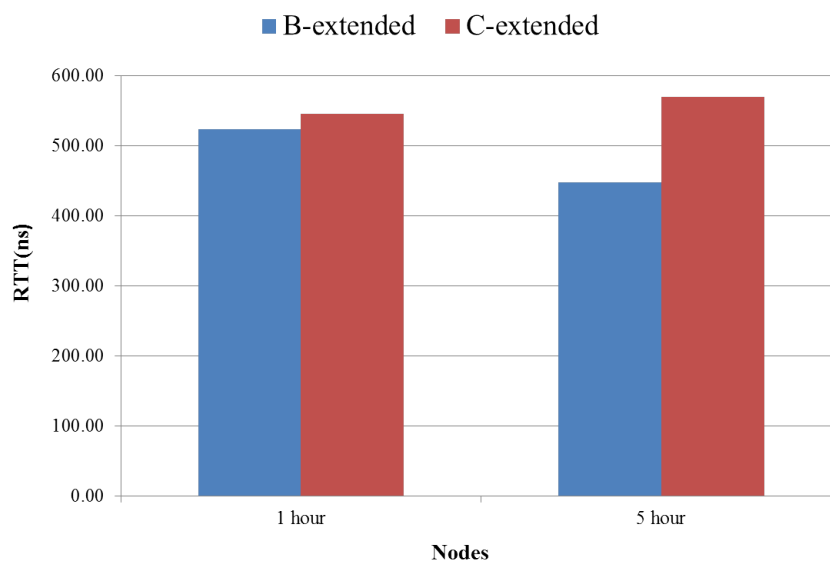


Figure 5.19. The controller setup for the extended experiments.

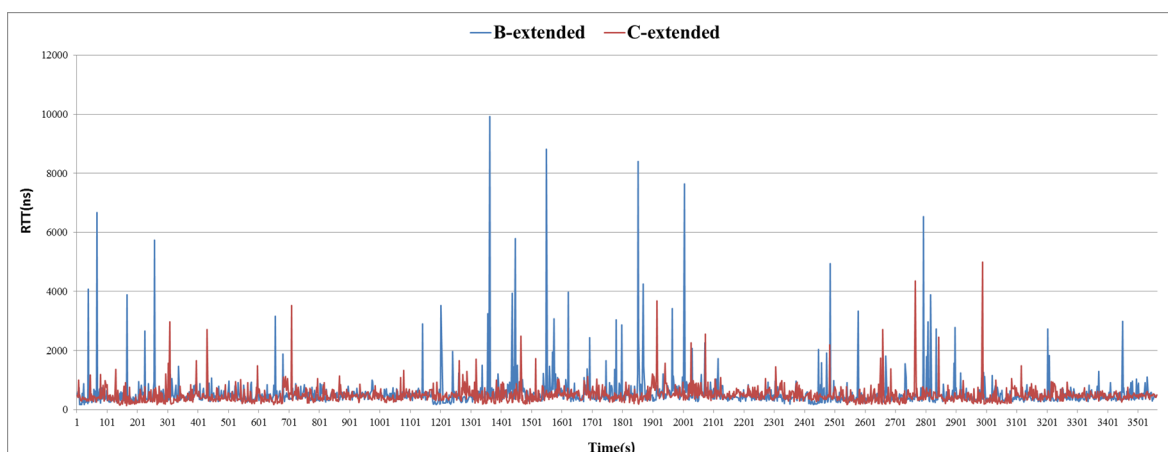


Figure 5.20. Average RTT for the extended experiments with 39 nodes for one hour.

The extended experiments labelled experiment *B-extended* for the distributed controllers using ONOS original and *C-extended* for the distributed controllers using fragmentation, were only on the distribution, because the central controller had repeatedly shown that it lagged behind the distributed settings

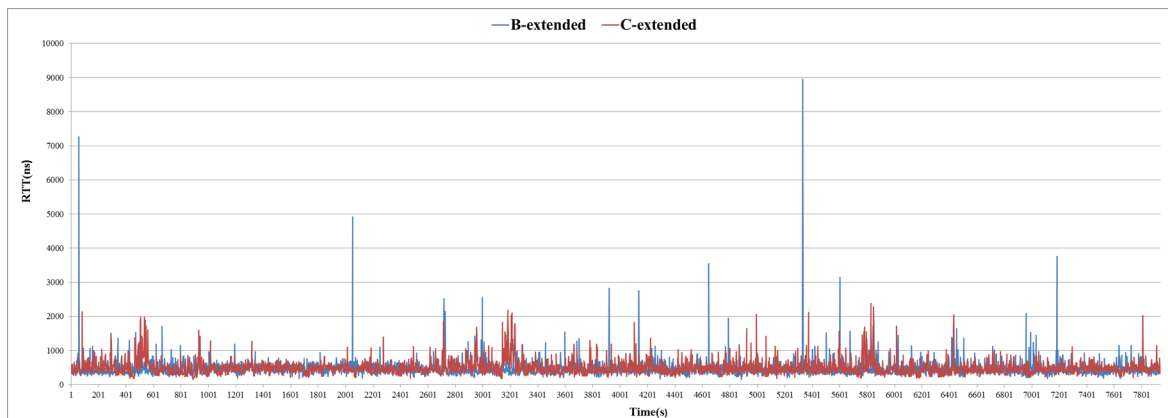


Figure 5.21. Average RTT for the extended experiments with 39 nodes for five hours.

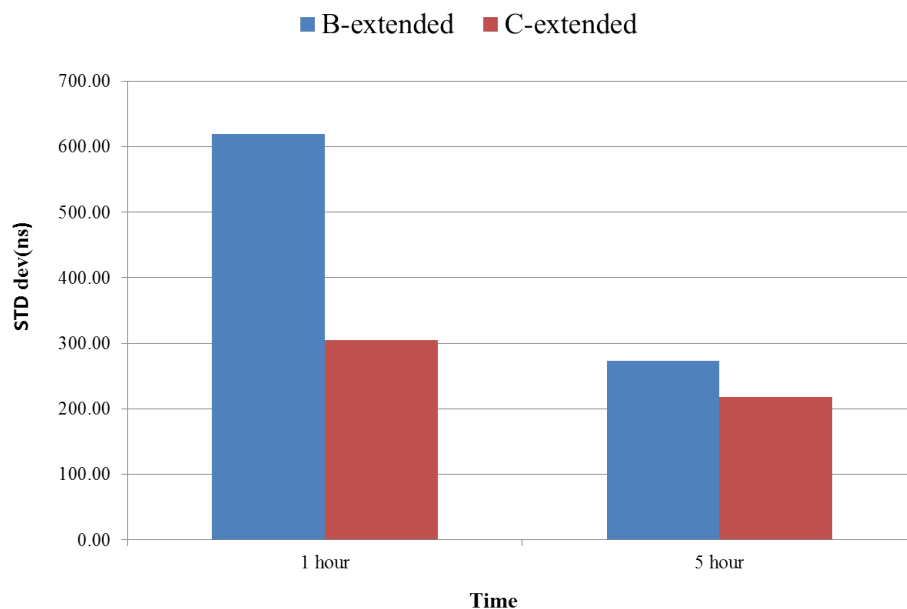


Figure 5.22. The standard deviation for the extended experiments.

and had also proved not to be ideal for the future utilisation of SDWSN. This was to stretch the two distributed experiments to show how they behave comparatively on an extended duration of the test. The results of the extended experiments refer to Figures 5.19 to 5.23.

For the controller setup time, the results are relative between the two experiments with the fragmentation slightly higher. The results are the same even for experiments *B* and *C*. Therefore, the ONOS-distributed setup performs slightly better as far as the connection time is concerned. This is caused by the two-level control structure in the fragmentation model. The fraction of the difference between the two

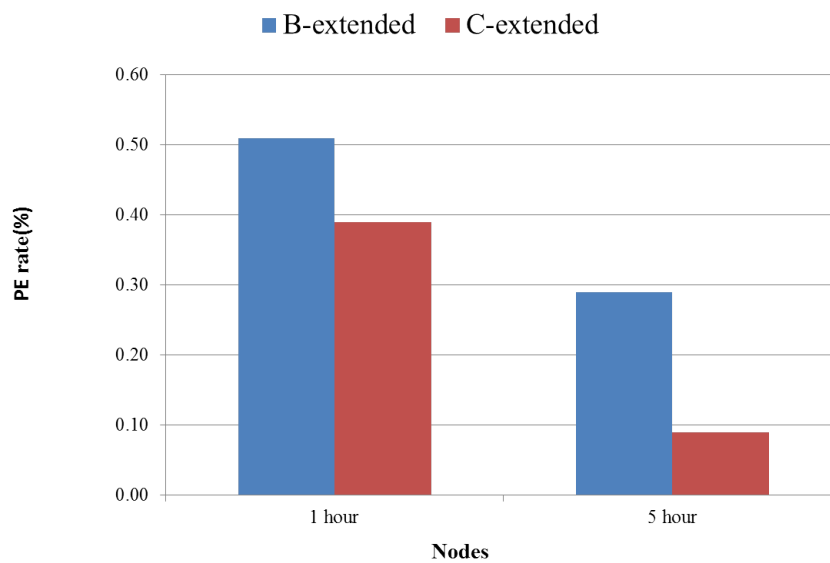


Figure 5.23. The packet error rate for the extended experiments.

is, however, not substantive enough to have any negative impact, and falls relatively within the same margins. The extended testing period cannot account for this because these are the first packets to be sent to the controller.

The RTTs observed are depicted in Figures 5.20 and 5.21 for scenarios one hour and five hours respectively. As in experiments *B* and *C*, the averages show the fragmentation model slightly above the original ONOS distributed but as shown in the graphs, the fragmentation performed better in terms of consistency for the whole testing periods. Here again, the disparity is caused by the number of packets produced in the fragmentation model which were again more in these tests. This is also apparent in the maximum RTTs; the fragmentation model consistently recorded the lowest. The fluctuating RTTs recorded in the ONOS distributed experiment cannot be good for optimal efficiency of the network. This inconsistency in the maximum RTTs is clearly shown by the standard deviation, which is high as shown in Figure 5.22. The standard deviation is very low in the five hour scenario in both experiments, which shows that the system normalises to consistency over time. The improvement brought by the fragmentation model is also apparent in this case. The packet error rate for the extended experiments was also standard and unlike in the other experiments, it was below 0.5%. The lowest rate recorded was in the fragmentation model for the five hour scenario. However, as stated in Subsection 5.2.5 above, the loss only occurs on the second sink node.

The results have shown that the extension of times produces approximately the same results as the other times do. Although the improvement of the fragmentation model is glaring in the extended-time experiments, the behaviour or the consistency is the same. However, there is a notable difference in the performance of the fragmentation model from the 24 to 39 sensor nodes. This shows that the model does not suffer performance deterioration under increased load or extended times. However, the scalability assertion needs further evaluation for certainty. The scalability could not be tested beyond this due to the limitations of the simulation tool.

5.3 CHALLENGES

Several challenges were encountered in the evaluation of the experiment. The SDWSN is a very new platform and therefore there are fewer supporting tools which could be used to prove concepts or build prototypes. The SDWSN consists mainly of the SDWSN network (SDN-enabled wireless sensor network) and an SDWSN controller. Most of the SDN controllers are not customised for SDWSN and likewise there are fewer SDN-enabled sensor nodes or simulation tools. The study is based on or built on the SDN-WISE solution which consists of an ONOS controller customised for SDWSN and a Cooja adaptation of SDN-enabled sensor and sink nodes. The proposed model was implemented into ONOS. This is ONOS 1.0.2 which is older than the current stable version in 1.9.0. The current version uses Hazelcast, which contributes to the packet loss suffered on the second controller of the cluster. In the SDWSN simulation, there was a limit to the number of nodes to simulate. More nodes crashed the system. This limited the extent of the scalability length this study desired to explore. Also, the address of the controller had to be entered promptly and manually for each sink node. Although this did not affect the material output of the experiment, an improvement would assist. The simulation could also be enhanced to enable other metrics to be tested. These challenges did not change the experimental outcome but could have enhanced the overall usability.

CHAPTER 6 EFFICIENT CONTROLLER PLACEMENT AND MASTER NODE RE-ELECTION

The fragmentation model is earmarked to play a huge role in stimulating participation of SDWSN in IoT. To realise this, we optimise the model for integration and operational efficiency. We consider two aspects: controller placement and controller re-election after failure. This chapter discusses the controller placement problem and the controller re-election for device mastership in the context of SDWSN. The rest of the chapter is organised as follows: In Section 6.1, we discuss the controller placement in SDWSN. This section includes a brief background, brief literature, controller placement in SDWSN, problem statement, a proposed controller placement solution for SDWSN, experimental evaluations, and results and discussion. This is followed by the controller re-election mechanism in a case of controller failure in Section 6.2, which also includes different subsections, namely background, current controller re-election methods, problem statement, proposed solution, experimental evaluation, and results and discussion. We conclude the chapter in Section 6.3.

6.1 THE CONTROLLER PLACEMENT IN SDWSN

The fragmentation model fragments the control logic so that each fragment controls a particular portion of the network. This, among other things, reduces the distance between the sink nodes and the controller and thus ensures that the network latency is minimised. In a large and operational network, the placement of these local controllers should be diligent and systematic; this is referred to as the controller placement problem.

6.1.1 Brief Background

The controller placement problem is a concept chiefly studied in the traditional SDN space. Its fundamental purpose is rooted in minimising latency between the nodes (devices or switches) and controller(s). This problem is, however, unexplored in the SDWSN paradigm. Its purpose is also to minimise the latency between the sink nodes and the controllers to ensure optimal performance. The overall and dominant conclusion from the existing controller placement work is that there is no recipe [209] nor any placement rules [210] that apply to all networks, but they offer guidelines through which optimal placement can be found. Therefore, there is no single best controller placement solution, especially when several performance and resilient metrics are used. There is only a trade-off between these metrics [209,210]. Heller *et al.* [210] further state that the placement problem has been well explored and no new theoretical insights are to be expected.

6.1.2 Brief Literature

This problem has been largely studied in the traditional SDN networks but not in the SDWSN. Heller *et al.* [210] were the first proponents of this problem which revolved around answering the two questions:

1. How many controllers are needed?
2. Where in the topology should they go?

These are two fundamental questions that need to be answered and they apply to SDWSN too. Heller's work presents comprehensive guidelines for operators to deploy multiple controllers effectively in their SDN networks. This research work set the scene for this problem. The work was aimed at determining the number of controllers to deploy and where in the topology they ought to be located. The authors determined that the controller placement problem was an NP-hard problem, at least as hard as the hardest problem. Nondeterministic Polynomial is a set of computational problems that can be verified in a polynomial time by a deterministic Turing Machine. Heller *et al.* [210] optimised the controller placement problem based on the latency metric and they developed two latency cases. The first is referred to as average latency which, is modelled as:

Given a network graph $G(V, E)$, let the edge weights represent propagation latencies, $d(v, s)$ represent the shortest path from node $v \in V$ to $s \in V$, $n = |V|$ is the number of nodes, S will be the number of controllers to choose from, S' is the number of controllers to be placed such $|S'| = K$, the average latency for a placement of controllers $L_{avg}(S')$ is modelled as:

$$L_{avg}(S') = \frac{1}{n} \sum_{v \in V} \min_{s \in S'} d(v, s) \quad (6.1)$$

Equation 6.2 represents the worst-case latency, which is modelled as:

$$L_{wc}(S') = \max_{v \in V} \min_{s \in S'} d(v, s) \quad (6.2)$$

The average latency uses a k-median [211, 212] optimisation approach. The k-median is a clustering method which seeks to find k-cluster centres such that the sum of the distances between the centre and all other points in the cluster are minimised. The k-median strives to minimise the 1-norm distances between each point and its centre in the cluster. The k-median was borne as an improvement of the much-studied k-means approach which also minimises the distance between the centre and the cluster points. The main difference is the variables they used; according to the names, k-median uses the median while k-means uses the mean. The reason behind the k-median is that k-mean is vulnerable to outliers [211]. The worst-case latency metrics by Heller *et al.* [210] uses the k-center clustering technique. The k-center aims to minimise the maximum distance between the centroid and the points in the cluster [213], thus it minimises the n-norm distance. The authors further deduced that one controller is sufficient to fulfil the existing reaction-time requirements although not fault tolerance.

Hock *et al.* [209] extend this work by adding resilience requirements, thereby showing that in instances where a single controller complies with the latency condition, more controllers would be necessary for the resilience requirement. In addition to controller failure, the authors consider intercontroller latency, network disruption, and load balancing. Hock's work is formally referred to as Pareto-based Optimal Controller placement(POCO) [209, 214]. The POCO framework is available on opensource. The POCO uses the worst-case latency from Heller *et al.* [210] because they argue that the average does not consider the worst-case scenarios. They further distinguish between a failure-free scenario and a controller failure scenario. A failure-free scenario is the optimisation where failure of the controller nodes is less, whilst the controller failure scenario occurs when the controller failure is expected to reach $k-1$, thus all fails but 1. Hock's failure-free latency metric is modelled as in (6.2), for distinction purposes we shall refer to it as the maximum latency for failure-free optimisation:

$$L_{ff}(S') = \max_{v \in V} \min_{s \in S'} d(v, s) \quad (6.3)$$

The controller failure optimisation, which is for the worst-case optimisation, is referred herein as maximum latency for controller failure optimisation and modelled as:

$$L_{cf}(S') = \max_{v \in V} \max_{s \in S'} d(v, s) \quad (6.4)$$

The controller failure effectively doubles the optimisation, and thus ensures that the maximum latency value covers all other deployed controllers k such that any k -1 controller failure is catered for. The first case in (6.3) equally distributes the controllers across the breadth of the network, while the second case in (6.4) moves all controllers towards the centre of the network, ensuring that in a worst-case scenario, the latency remains within an acceptable range. The authors conclude that the first scenario suffers from high latencies in worst-case scenarios but lower latencies in failure-free scenarios whilst the second performs better in the worst-case scenarios and worst in failure-free scenarios. This work also investigated the inter-controller latency and load balancing.

This work is extended in [215] by using heuristics methods to cater for large-scale and dynamic networks. Bari *et al.* [216] propose a dynamic provisioning of controllers according to their activeness in the network. Lin *et al.* [217] propose a controller placement algorithm that aims to determine the number of controllers and their location on the network. They also prove that controller placement is an NP-completeness problem. The work of other researchers such as Ishigaki *et al.* [218] and Muqaddas *et al.* [219] focuses on reducing the inter-controller latency instead of the node/switch-to-controller latency.

The closest to SDWSN is the work by Reze *et al.* [220] which deals with the efficient deployment of multi-sink and multi-controllers in WSN for a smart factory. The authors optimise the placement problem as integer linear programming (ILP) which ensures that every sensor node is covered by at least one sink node and at least one controller node. The distance estimation used Dijkstra's shortest-path algorithm.

6.1.3 Controller placement in SDWSN

Most of the research works above are based on the traditional SDN, except [220]. However, [220] deals with the placement of the sink nodes and the multi-controller placement, which follows the conventional way of distribution, against which this research work argues. Overall, the controller placement footprint is lagging behind in SDWSN. However, as stated in the previous chapters, SDN

solutions provide insightful guidelines for developing SDWSN solutions. The only major distinction is in the varying traits of the networks, with SDWSN typified by limited resources. Therefore, most SDN solutions need to be customised to work in SDWSN. The controller placement problem in SDWSN is vital for optimal performance and efficiency of the network. In traditional SDN, the problem seeks to reduce the latency between the SDN switches and the SDN controllers. However, in SDWSN, particularly our solution, it is between the sink nodes and the controllers. As observed in Chapter 5, the two-level architecture adds a little latency; hence, the challenge is to ensure that this latency is kept at a very minimum to insignificant levels.

The challenge is to ensure that the placement of both the local controllers and the global controller(s) follows a procedure that will minimise latency and provide resilience. This would also determine the number of local controllers for any given network. The placement of the global controller is also especially important to minimise the latency to and from the local controllers. In most situations, the controller placement problem entails controller failure, network disruption, load balancing, and inter-controller latency [209]. The fragmentation model deals with most of these problems. The inter-controller problem is out because there are no data exchanges between the local controllers, except with the global controller, in which case, the latency does not affect the convergence of data. Network interruptions are rife in SDWSNs but do not affect the operation of the control logic, otherwise the latency between the sink nodes and the controller is minimised by bringing the controller service closer.

6.1.4 Proposed controller placement

The purpose of this exercise is to optimise the fragmentation model with the aim of minimising latency between the sink nodes and the local controllers, as well as between the local controllers and the global controller. Unlike the solutions reviewed above, this focuses on the sink-to-controller latency. We assume that the placement of the sensor nodes and sink nodes is in accordance with the demands of the network or the service being provided. Therefore, the control framework needs to respond to the needs of the network, not the other way around.

As stated in Heller *et al.* [210] the controller placement problem has been studied extensively and no new theoretical framework is likely. The theoretical framework as proposed by Heller *et al.* [210] and

further by Hock *et al.* [209] provides a fundamental baseline. We apply this in the SDWSN, particularly for the fragmentation model. Heller *et al.* [210] provided two use-case scenarios, the average case which uses k-median and the worst case which uses k-center. These optimisation techniques are applicable to SDWSN. We adopt the average-case scenario by Heller using k-means. Although Heller states that this formula is for k-median, it actually applies to both because it produces the mean and the median, as well as the maximum value used in k-center; it therefore depends on the use and choice of the criterion. We use this to optimise for latency, we deal with resilience in Section 6.2. The method uses a k-means clustering approach which minimises the distance between the node (sink node) and the cluster centre (controller). This will ensure that the latency between all sink nodes and the controller is minimised. This is modelled as in (6.1); the fragmentation latency L_{frag} is modelled as:

$$L_{frag}(S') = L_{avg}(S') = \frac{1}{n} \sum_{v \in V} \min_{s \in S'} d(v, s) \quad (6.5)$$

The k-means clustering partition data observations of any n-by-p matrix into k-clusters, it returns an n-by-1 vector consisting of the cluster centres called centroids [221]. K-means uses different distance metrics to compute the $d(v, s)$ such as Euclidean (default), Cosine, Cityblock, Hamming, and Correlation. It uses a k-mean++ algorithm for choosing the best k , which is an improvement on the original criterion which often led to poor clustering.

The latency threshold in SDWSNs should be much lower than it can be in traditional SDN, owing to the lower data rates of the sink nodes. This will augur well for the fragmentation model, which seeks to bring control logic closer to the nodes. The placement of the global controller is also important, and therefore, we use the second latency optimisation, the worst case as proposed in [209]. This entails minimising the latency between the local controllers and the global controller(s). This effectively doubles the optimisation and locates the global controller at the centre of the entire network. The formula follows that in (6.4) which is modelled as latency for the global controller L_{gc} :

$$L_{gc}(S') = \max_{v \in V} \max_{s \in S'} d(v, s) \quad (6.6)$$

6.1.5 Experiment

We use Matlab to model the controller placement. Matlab is rich in tools used for algorithm testing and mathematical modelling. Accordingly, there are built-in tools for clustering. The Matlab built-in k-means uses a k-means++ algorithm. The formula used in Matlab is [221]:

$$[idx, C, sumd, D] = kmean(X, K, Name, Value) \quad (6.7)$$

where:

- X is the matrix under observation i.e. graph G above.
- K is the specified number of clusters required.
- The name-and-value pair represents the distance metric and its name i.e. 'Distance', 'cityblock'.
- idx returns a vector which contains the cluster indices of observation matrix X (graph).
- C returns the k -cluster centroid locations, these are the centres of the partitioned clusters.
- $sumd$ is a k -by-1 vector which contains the within-cluster sums of each point to cluster distance in the cluster.
- D returns a vector of n -by- k matrix of all distances from each point to every centroid.

Matlab simulation defines nodes with no specification of the type. The set of nodes in our experiment represents the sink nodes amongst which we want to place the k number of controllers. We placed fifteen (15) sink nodes at different locations to determine the locations of the three (3) local controllers and the global controller. We used actual latitude and longitude coordinates as listed in Table 6.1 below. Table 6.2 lists the weights of the coordinates using the Haversine distance method. The idea is also to affirm the distances using k-means built-in distance metrics such as Euclidean and Cityblock. The different distance calculation methods returns approximately the same results on short distances. We first run the k-means algorithm in Matlab to determine the locations of the local controllers, then use those locations to determine the location of the global controller.

Table 6.1. Sink nodes with coordinates

Controller	Coordinates	
	Latitude	Longitude
1	-25.755584	28.278443
2	-25.755594	28.278934
3	-25.755668	28.278703
4	-25.756051	28.27778
5	-25.756131	28.278107
6	-25.756303	28.278242
7	-25.756447	28.279639
8	-25.756334	28.279703
9	-25.756331	28.279551
10	-25.756536	28.27827
11	-25.756528	28.278532
12	-25.756207	28.280012
13	-25.756572	28.280249
14	-25.755309	28.278177
15	-25.755426	28.278749

Table 6.2. Coordinates and weights/distances

Edge Connection	Weight(m)
(1,2)	49
(1,3)	28
(2,3)	25
(4,5)	34
(4,6)	54
(5,6)	23
(7,8)	14
(7,9)	16
(8,9)	15
(4,10)	73
(5,10)	48
(6,10)	26
(4,11)	92
(5,11)	61
(6,11)	38
(10,11)	26
(7,12)	46
(8,12)	34
(9,12)	48
(7,13)	63
(8,13)	61
(9,13)	75
(12,13)	47
(1,14)	41
(2,14)	82
(3,14)	66
(1,15)	35
(2,15)	26
(3,15)	27

6.1.6 Results and Discussion

The purpose of this exercise was to determine the best locations to place the local controllers and the global controller for the fragmentation model. The k-means method produces, in addition to average latency, the maximum latency (referred to by Heller *et al.* [210] as worst-case latency) and the distances of each point to the centroid. The locations of the three local controllers based on the k-means are depicted in Figure 6.1 and listed in Table 6.3.

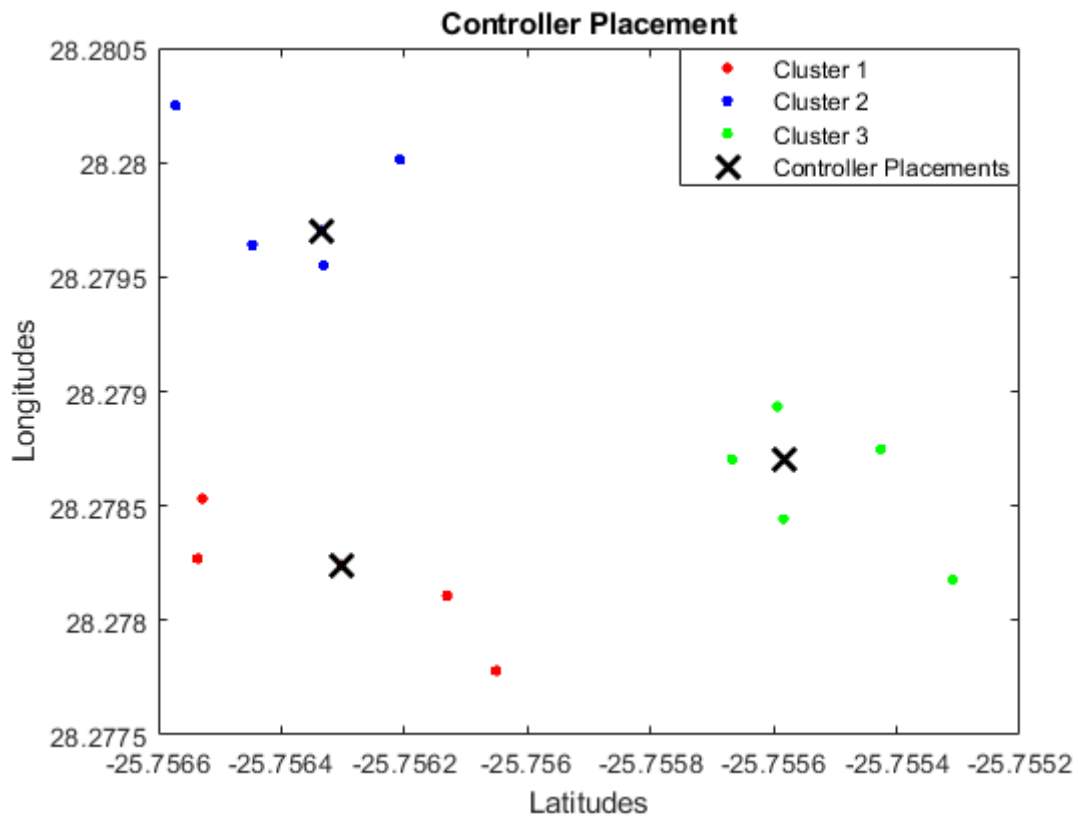


Figure 6.1. The locations of the local controllers by k-means algorithm.

Table 6.3. The locations of the local controllers

Controller	Coordinates	
	Latitude	Longitude
A	-25.7563	28.2782
B	-25.7563	28.2797
C	-25.7556	28.2787

The main antagonists of the k-means algorithm argue that the placement is vulnerable to outliers, and as seen in Figure 6.1, the local controllers are placed closer to or on one of the sink nodes. Moreover, it is said that this poses a challenge in a very large network where distance in between are huge. But as in the case of the SDWSN, particularly the fragmentation model, this does not have a negative effect because the network is already fragmented. This is the reason we chose k-means, and furthermore, in a relatively small network such as the fragmented clusters, the different clustering techniques produce relatively similar results. This can be observed from the average value, the median value, and the maximum value produced.

On the placement of the global controller, we applied the k-means to the locations obtained when placing the local controllers above. This is highlighted in Figure 6.2 below. As shown on the figure, the global controller is placed closer to the local controller closest to the mean.

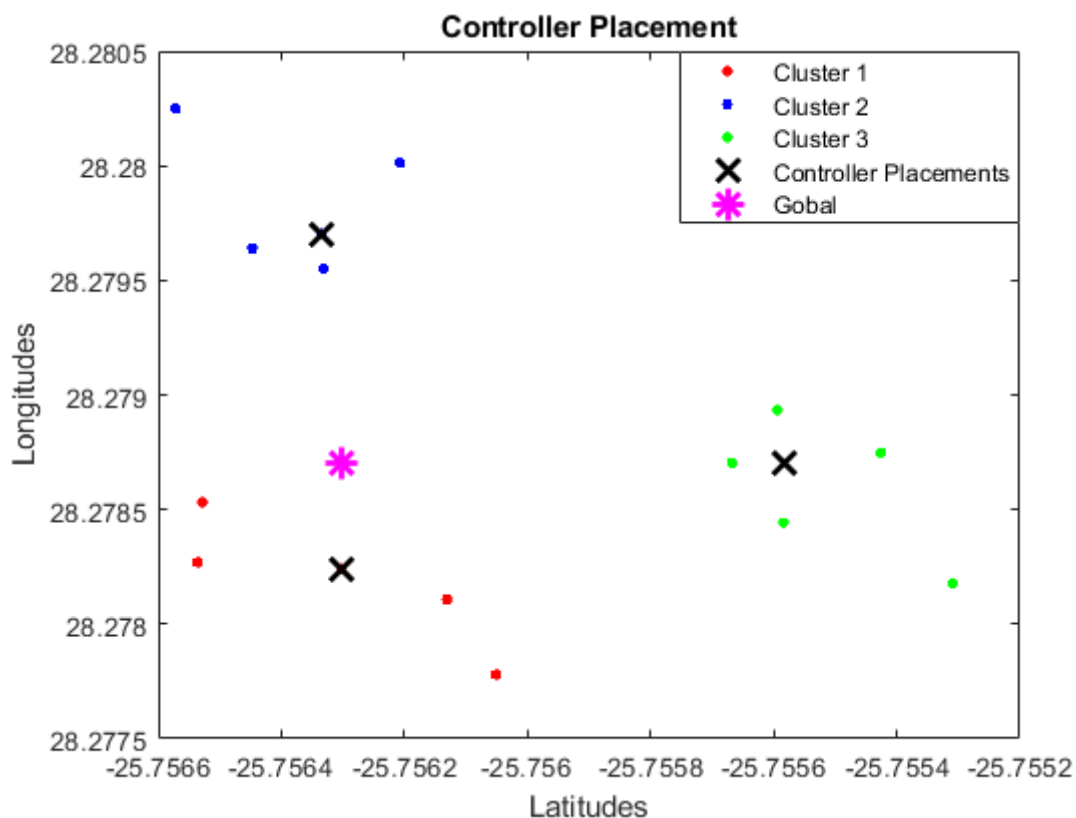


Figure 6.2. The locations of the global controllers by k-means algorithm.

Unlike the local controller placement, the global controller is expected to be at a relative distance to the local controllers and therefore placement closer to the mean would leave other observations vulnerable

to high latencies. Therefore, we re-optimize the local controller locations to counter the error condition. The use of k-median is considered. However, it results in the same placement as the mean, as shown in Figure 6.3.

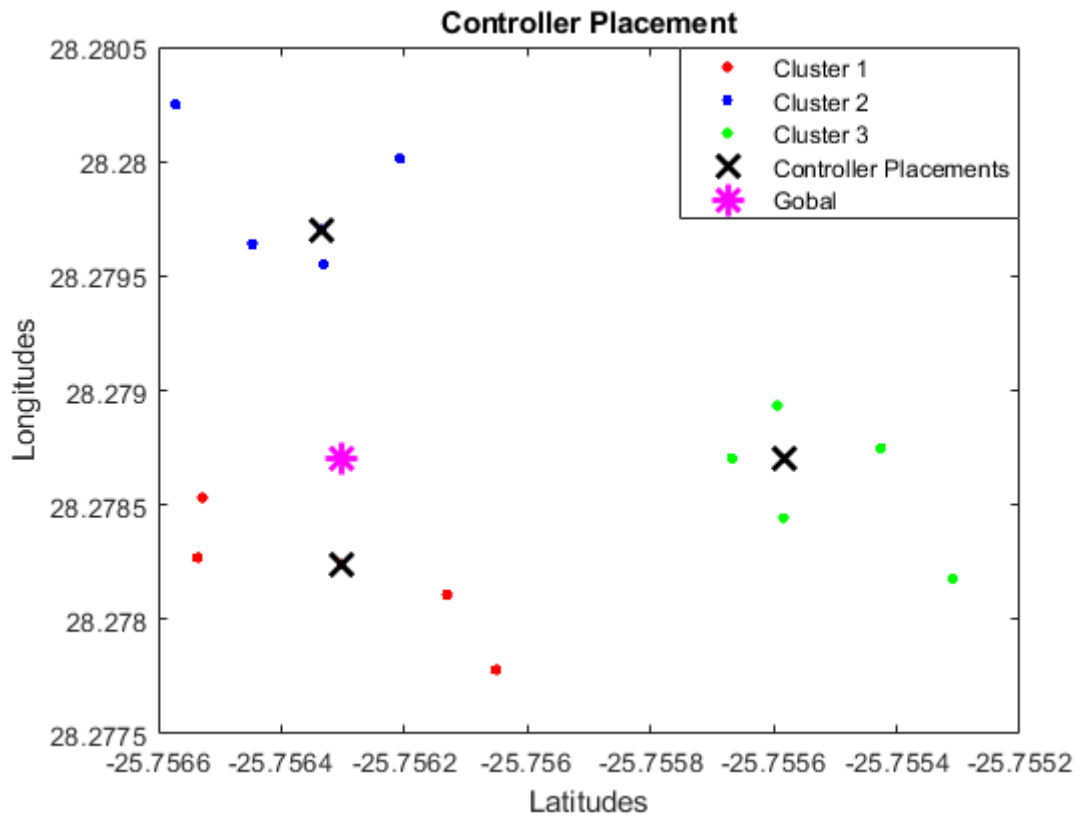
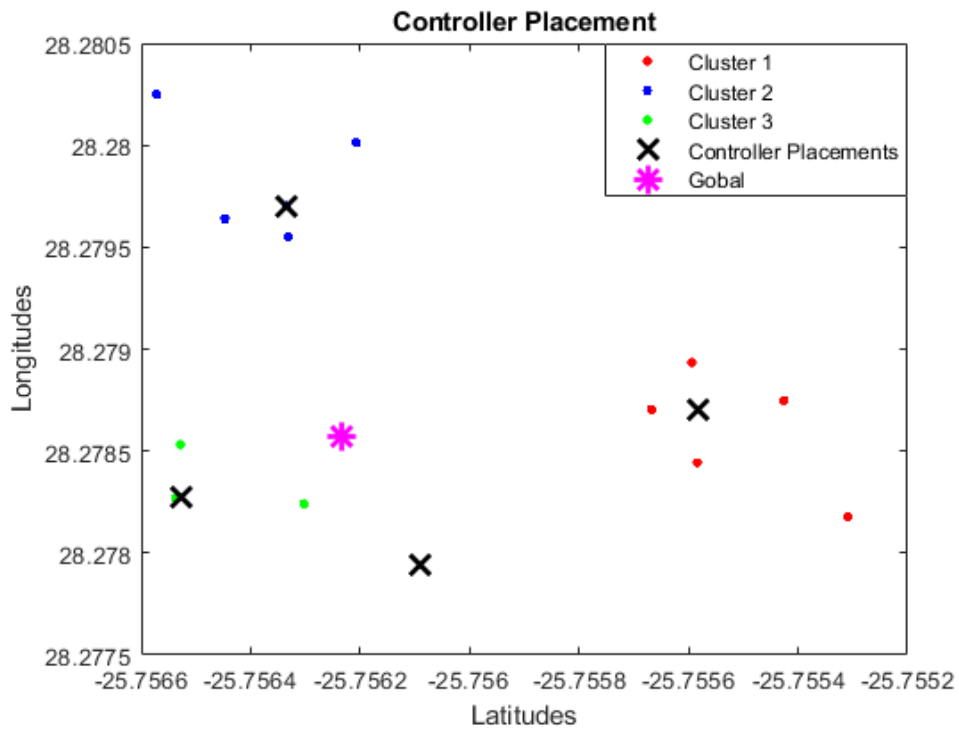
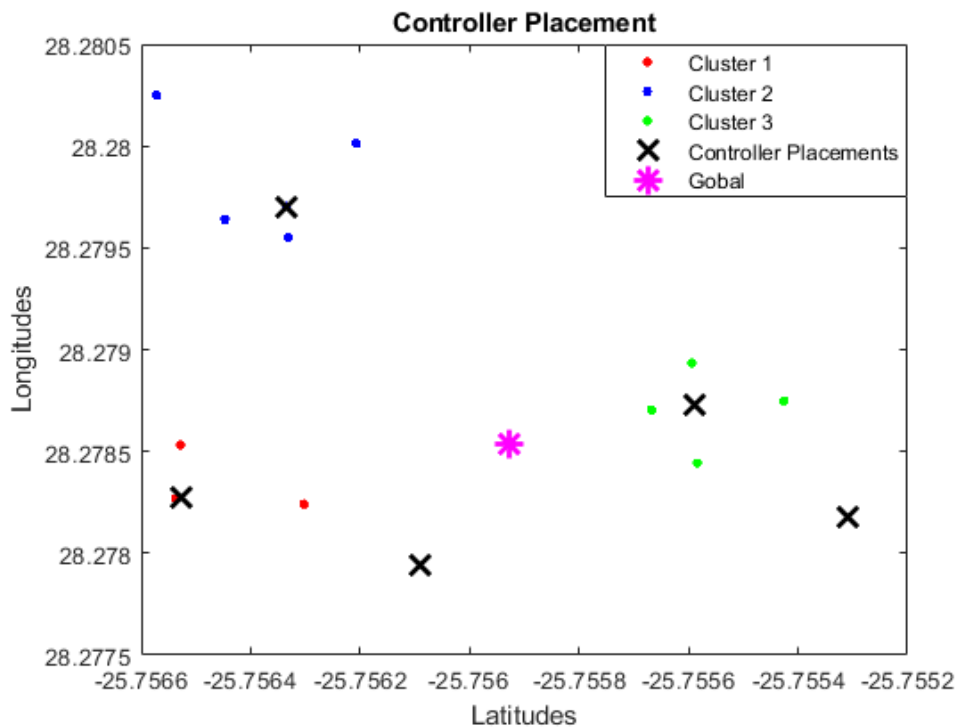


Figure 6.3. The use of the median is the same as that of the mean.

Increasing the topology from three to four or five local controllers (in case the topology grows) does not improve the positioning either. This means that the optimal placement of the global controller cannot rely on the increase in topology (data points) as shown in Figures 6.4(a) and 6.4(b). The mean would always be skewed and unpredictable. Besides the fact that this does not have much effect on centralising the global controller, this would be a bad method as it is not cost effective.



(a) The global controller placement with $k = 4$.



(b) The global controller placement with $k = 5$.

Figure 6.4. Large topology does not change the placement of the global controller when using k-means.

Therefore, we used the concept of moving-median to re-optimize the locations of the local controllers to determine the optimal location of the global controller. Moving-median computes a number of different k medians over a sliding window of the size of k . It shifts the window forward and backward over the size of k . It is mostly used to compute time series data. Table 6.4 lists the new locations derived from the moving median. The next step is to apply k-means to this new matrix to determine the new location of the global controller; we also find the median.

Table 6.4. The locations of the local controllers after the moving median function

Controller	Coordinates	
	Latitude	Longitude
A	-25.7563	28.2790
B	-25.7563	28.2787
C	-25.7560	28.2792

The new matrix above mimics the worst-case scenario described by Heller while conforming to the free fail scenario by Hock. These new locations move towards the k-center optimisation. However, noticeably, they cannot be used for the local controller placement as they converge towards the centre of the network. Figure 6.5 shows the locations of the local controller using the new matrix. Evidently, this is not suitable for the local controller placements according to the fragmentation ideals for SDWSN because as shown in Figure 6.5, they are further away from the sink nodes, which are their primary area of interest.

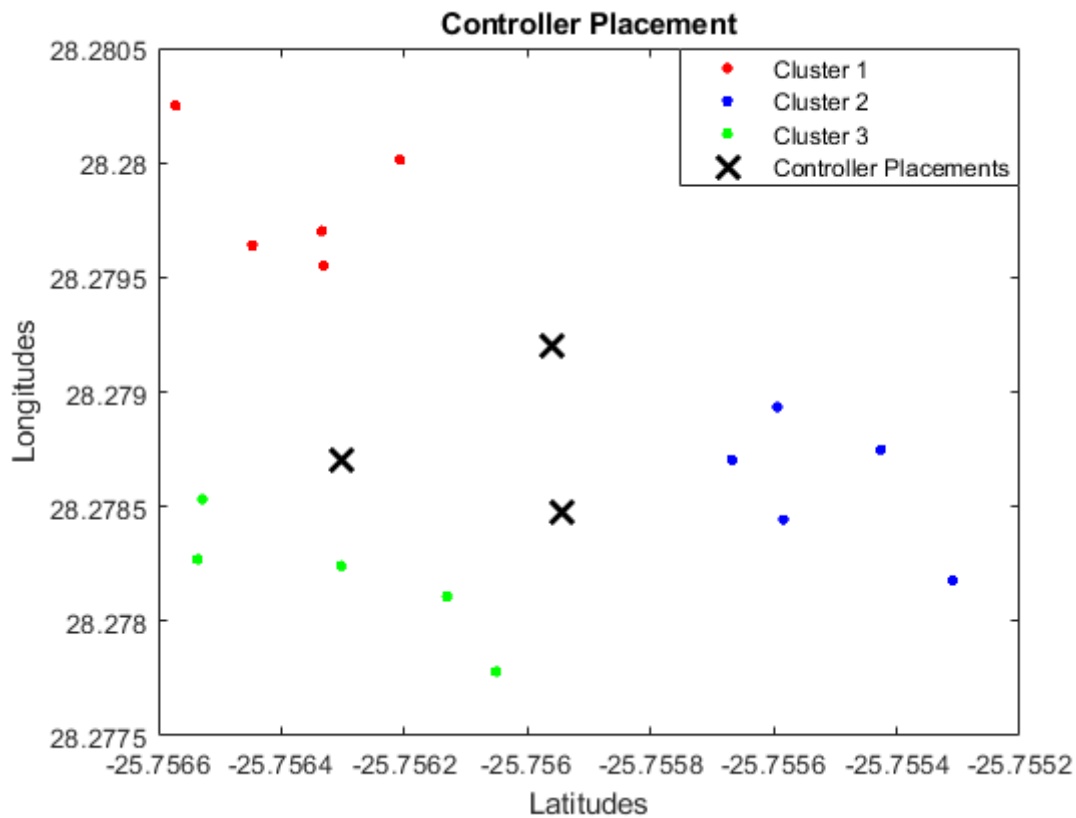
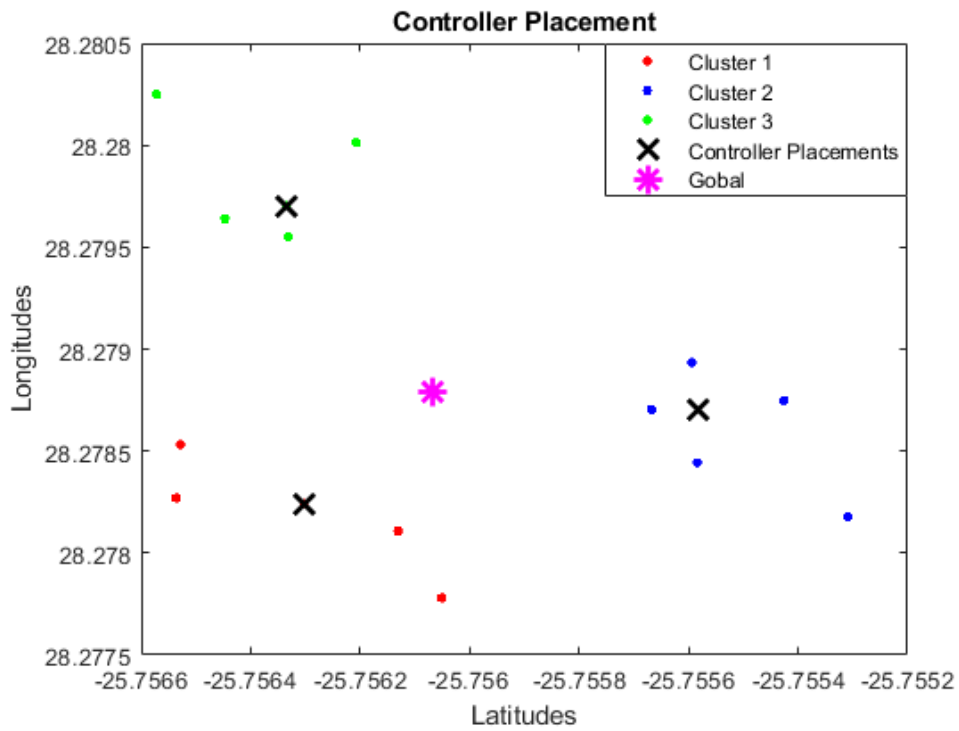
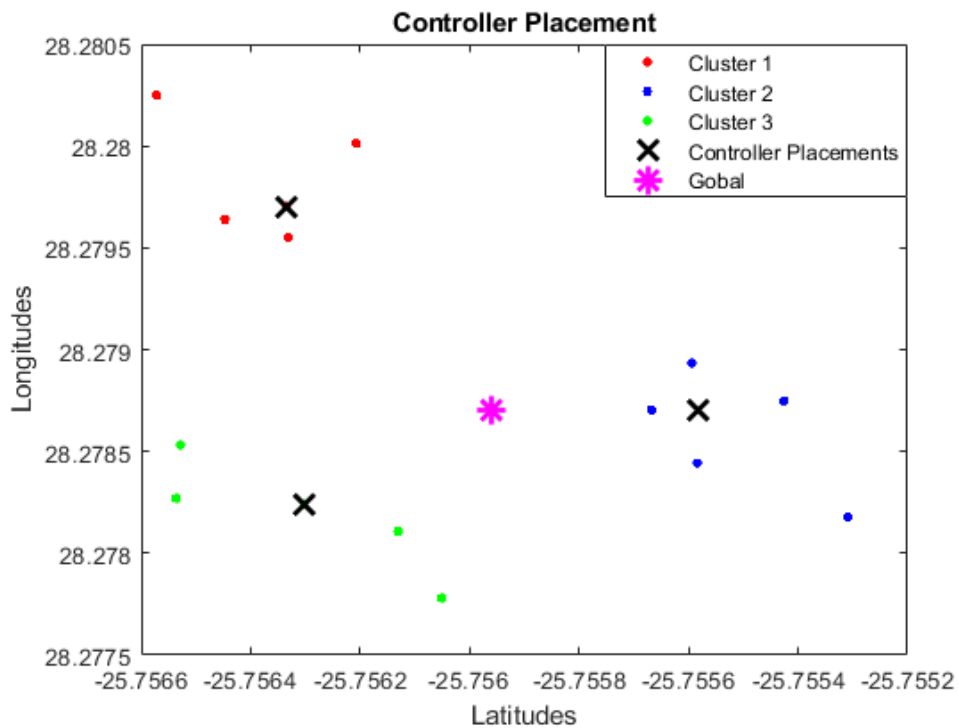


Figure 6.5. The locations of the local controller after re-optimisation.

However, as these locations converge towards the centre of the network, they improve the placement location of the global controller. Figure 6.6 shows the location of the global controller with the mean and the median respectively. This results from applying the mean and median to the three locations obtained from the moving median. The placement of the global controller improved greatly. There is a slight difference between the location of the mean and that of the median.



(a) The mean of the moving median locations.



(b) The median of the moving median locations.

Figure 6.6. The final optimisation of the locations to determine the global controller location.

To find a better method between the mean and the median, we looked at the distances to each point. We took the cluster locations in Tables 6.3 and 6.4 to determine the distance between each point and the centroid (global controller location) produced by the mean and median of the first matrix (Table 6.3) and the mean and the median of the moving median matrix (Table 6.4). We referred to the first matrix as S and the second matrix as S' . The mean and the median of the first matrix(S) were the same. We used the Haversine method to calculate these distances using a different platform (Java code). The results are captured in Table 6.5.

Table 6.5. The distances between obtained locations and the centroids

Location	Distances (m)		
	S : mean/median	S' :median	S' :mean
A	78	83	73
B	100	70	71
C	50	80	81

Table 6.5 shows that the mean has a better overall distance although the difference is slight. Both the mean and the median could therefore be used to determine the final location of the global controller. We can therefore deduct that the k-mean is suitable to place the local controller while the global controller should be placed by further optimising the local controller locations. This eventually converges to the k-center approach. The following Algorithm 6 summarises the steps of placing the controller.

Algorithm 6 Controller placement

These are the steps to be followed:

- Place the local controllers using k-means or m-median clustering.
 - Use the locations of the local controllers to determine the location of the global controller.
 - Apply a moving median to the locations of the local controllers.
 - Apply k-mean or k-median to the new derived locations to find the mean or the median for the location of the global controller.
-

6.1.7 Latency

The k-mean function in Matlab returns the best distances in the clusters. The average sums of the distances between the points in the cluster and the centroid (local controllers), measured in metres are 1.6, 1.6, and 1.8 for the three clusters respectively. The maximum distances between all the points to the centroids are 2.6, 2.5, and 2.3 in the three clusters. The average sums represent the k-mean value while the maximum points represent the k-center as described in Heller *et al.* [210] and Hock *et al.* [209]. The average distance between the local controllers and the global controller is 3.27. We use the standard distance, time, and speed equation to determine the latency in the form of time.

$$\begin{aligned} \text{Distance} &= \text{Speed} \times \text{Time} \\ \therefore \text{Time} &= \frac{\text{Distance}}{\text{Speed}} \end{aligned} \tag{6.8}$$

We use the Ethernet over copper speed of 197863.022 ms. The resultant latencies are 8.0864, 8.0864, and 9.0972 ns respectively within the local clusters and 16.5266 ns between the local controllers and the global controller.

6.2 CONTROLLER RE-ELECTION PROBLEM

Distributed systems use consensus algorithms to achieve state convergence. In Chapter 4, we discussed consensus algorithms in terms of replicating the state amongst the participating controller nodes. Another perspective of the consensus algorithms is the election of a leader in a distributed system. The election of the leader ensures reliability in the system in the event of failure. It ensures that for any operation, there will be service and backup for assurance. As in database systems, consensus is very important for SDN, especially in distributed SDN controllers.

6.2.1 Background

The golden rule in distributed SDN is that a device should be connected to at least one controller but can only be controlled by at most one controller at any time. The role of the consensus algorithm is to maintain this and to ensure that leadership changes are consistent, procedural, and efficient. Paxos [25, 26] and Raft [24, 222] are two of the most popular consensus algorithms used in SDN distributed controllers.

These algorithms are normally applied to in-memory data grid frameworks such as ZooKeeper [223], Hazelcast [207], and Atomix [208]. Both Zookeeper and Atomix use Raft as their consensus algorithm, while Hazelcast uses Best Effort and Anti-entropy. The ONOS framework used Hazelcast up to version 1.4, from which they started to use Atomix (because of the split-brain problem). Atomix uses a strong consistency data model, which is not suitable for the fragmentation model as explained in Chapter 4. The fragmentation model is based on the eventual consistency data model, thus eventual consistency ONOS over strong consistency ONOS. The Hazelcast framework also allows a choice between strong or eventual consistency.

6.2.2 ONOS device mastership

The mastership service in ONOS provides cluster management, synchronisation, and device mastership. The mastership service manages the device mastership. The ONOS device mastership management defines three roles [108, 224]:

- Master: The controller node has knowledge of the device and has full control.
- Standby: The node has knowledge of the device, and can read the state, but cannot control the device.
- None: The device may or may not have knowledge of the device and cannot interact with it.

This is the ONOS mastership life cycle. A controller node becomes a master of a device if it discovers the device first and can verify that the device has no master and has a control channel to the device. All other nodes that subsequently discover the device become Standby (if they have a connection) or None. The roles can change through administrative intervention, device disconnection, and disconnection from the cluster (split-brain syndrome). All these will prompt a role relinquishment and node re-election to replace the master. Re-election can be the result of master node failure, device disconnection, and an administrator intervention. The node relinquishing the responsibilities can elect a new master. The candidate to be the new master is selected from the standby nodes. The standby nodes are ordered on a preference list and the next node on the list becomes the candidate, master select

6.2.3 Proposed controller re-election

The fragmentation model distributes the controller instances across the network. In Section 6.1, we discussed the best ways to consider when placing the local controllers. The main metric behind the controller placement problem is latency, which is modelled through distance. Therefore, if the initial placement takes distance into consideration, then the re-election of the master should do so too. However, currently ONOS does not consider distance in its device master re-election. In the traditional SDN, this might not be a concern, considering the available infrastructure resources. The SDWSN, unfortunately, does not possess the luxury of resources to spare, hence more consideration should be taken into account.

The current implementation contradicts the aims and ideals of the fragmentation model. This section aims to optimise the controller master re-election method. This entails enhancing the method with a distance consideration when a new master node is elected. Building upon the controller placement; we implement the Haversine formula of distance. This calculates the distance based on the latitude and longitude coordinates of the controllers. We gather the coordinates of the controller instances upon commissioning. The pseudocode of the Haversine algorithm is given in Algorithm 7 below.

Algorithm 7 The Haversine algorithm

longitude lon1

longitude lon2

latitude lat1

latitude lat2

$R \leftarrow 6367000$ is the radius of the earth in metres.

$\Delta long \leftarrow lon2 - lon1$

$\Delta lat \leftarrow lat2 - lat1$

$\alpha \leftarrow (\sin(\frac{\Delta lat}{2}))^2 + \cos(lat1) \times \cos(lat2) \times (\sin(\frac{\Delta long}{2}))^2$

$\beta \leftarrow 2 \times \arctan(\sqrt{\alpha}, \sqrt{1 - \alpha})$

$\gamma \leftarrow R \times \beta$

return γ

The local controllers should be at the centre of their clusters according to the placement criteria discussed in the previous section. We calculate the distances between the current master controller

(the node relinquishing the role) and the standby controllers; the closest controller node becomes the candidate or the master select. The change of the master increases the latency, this exercise manages that increase by preferring the closest controller.

The standard procedure for controller re-election after a controller node failure in ONOS is described in Algorithm 8:

Algorithm 8 Controller mastership re-election procedure in ONOS

Upon a controller node failure:

- Relinquish the mastership role.
- The candidate node from the standby list is chosen as the replacement.

The standby list is populated as follows:

- First, the first controller to discover a device becomes the master controller, the rest become standby controllers for that device if they have a connection; they become none if they do not.
 - All standby controllers are stored on a list in a first-come precedence order.
 - The first controller on the list becomes the candidate.
 - Upon controller failure, the candidate controller becomes the master.
-

The proposed controller mastership re-election after failure is described in Algorithm 9 below.

Algorithm 9 The proposed controller mastership re-election procedure

Upon a controller node failure:

- Relinquish the mastership role.
- The candidate node from the standby list is chosen as the replacement.

The procedure for the proposed enhancement changes the standby such that:

- Uses a HashMap to store the standby controllers with their coordinates.
- Calculates the distance between the master and all the standby controllers, and store the results in a HashMap.
 - Calls the Haversine distance method.
- Selects the entry with the lowest distance as the candidate controller.
- Upon Failure, the candidate becomes the master.

Haversine distance calculation method:

- Find the latitude and longitude coordinates of both the current master controller and all the controllers in the cluster.
 - Measure the distance between the two coordinates.
 - **return** distance
-

The proposed method can be proactive or reactive. The proactive mode will ensure a fast transition from a failed controller to the replacement controller, but it will add overheads. Whereas the reactive method will be slower to transit to the replacement controller but with no processing overheads. Although the differences will be very minimal, the proactive method is more suitable for delay sensitive networks such as SDWSN.

6.2.4 Experiment

The controller master re-election enhancement was implemented on the ONOS mastership service. We set up a small experiment to test this enhancement. The experiment consisted of one global controller and three local controllers according to the fragmentation model (see the experiment in Section 5.1). The three local controllers consisted of 2GHz CPU and 1G RAM while the global controller consisted of 2GHz CPU and 2G RAM. The global controller also ran the simulation tool. The simulation consists of three sink nodes with ten sensor nodes each. All the controllers were virtual and placed in different

geographical buildings at the Council for Scientific and Industrial Research (CSIR), Pretoria. Figure 6.7 depicts the buildings where the controllers were located.

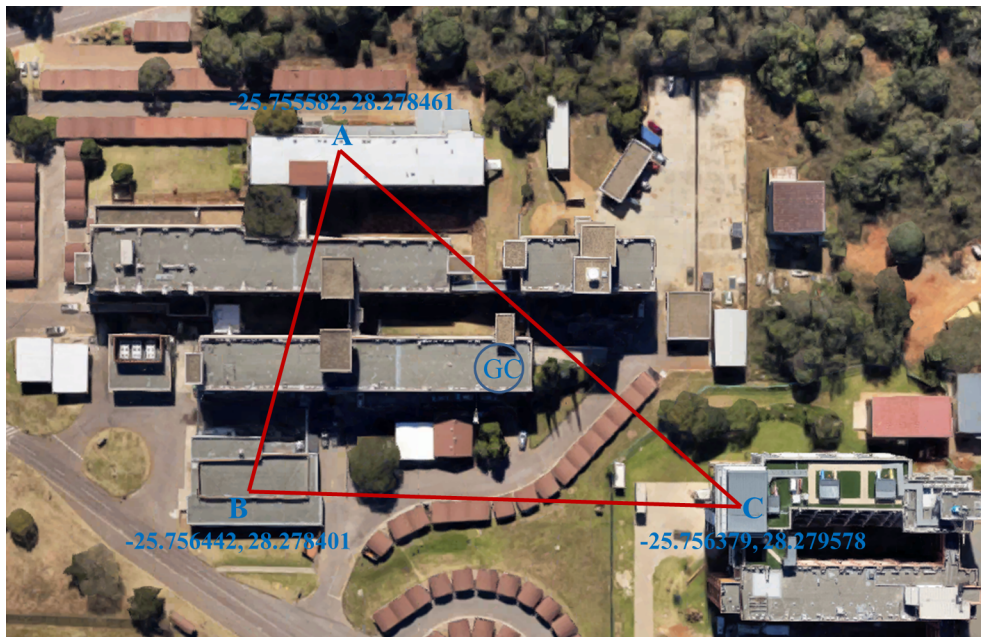


Figure 6.7. The location of the controllers in the buildings of CSIR, image from Google Maps.

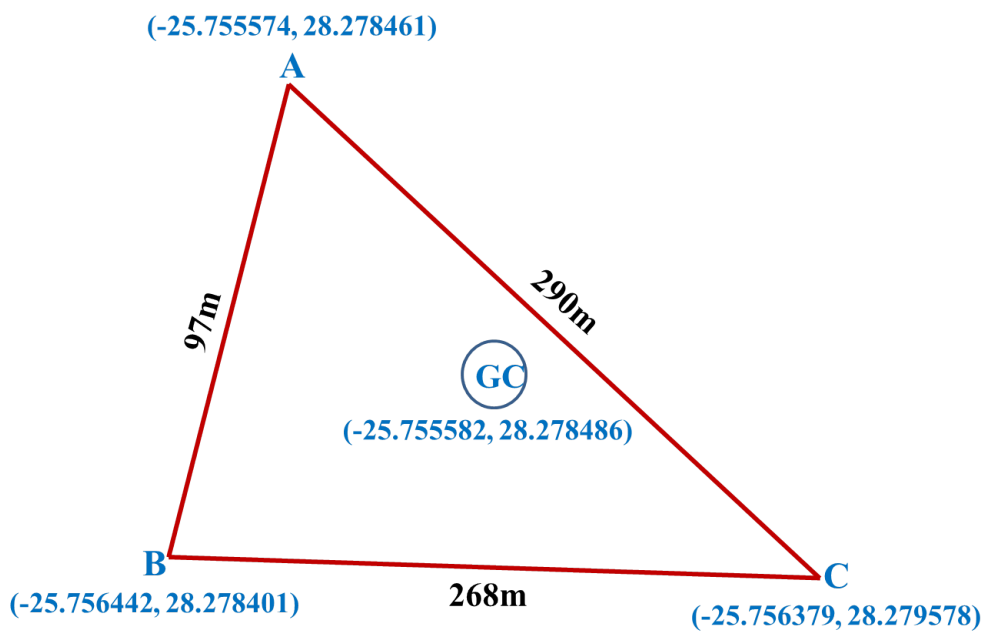


Figure 6.8. The geometric extraction of the controller locations.

The controllers were located in different buildings; Figure 6.8 shows the latitudes and longitudes of the controllers' locations, as well as the distances in between. The central controller was also located, more in the centre according to the placement model discussed in Section 6.1 above. The controllers and their location coordinates are summed in Table 6.6 below.

Table 6.6. The local controllers and their location coordinates in latitudes and longitudes

Controller	Coordinates	
	Latitude	Longitude
A	-25.755574	28.278461
B	-25.756442	28.278401
C	-25.756379	28.279578
GC	-25.755582	28.278486

We used a script provided by ONOS to manipulate the state of the controllers, we used "*onos-service*". This script allowed us to stop, start, restart, and check the status of a controller instance, thus "*onos stop/start/restart/status*". The testing procedure followed was:

1. Set up the cluster.
2. Verify that all controller instances are in ACTIVE state.
3. Run the SDWSN simulation.
4. Stop the one local controller instance in the middle of the simulation.
5. Verify if the stopped instance is indeed in INACTIVE state.
6. Confirm if the correct controller node was re-elected.

In the first experiment, we evaluated the proposed criterion by failing the local controllers one at a time. The second experiment centred around the current re-election criterion versus the proposed criterion in terms of latency. We measured the latency with the current criterion, then failed the controller and did the same with the proposed criterion. Controller B was used in this case. We measured latency on controller B, then failed it; then measured the latency with the replacement.

6.2.5 Results and Discussion

The ONOS framework provides a command-line service which allows monitoring and manipulation. The setup depicted in Figures 6.7 and 6.8 shows the different locations of the controllers with coordinates. The figures present a clear logical preference based on the different distances between the controllers, and thus the evaluation seeks to validate rather that test. The Haversine formula is also clear and we validated the formula away from ONOS to get the distances and used Google maps to verify them. The results obtained followed the logic as expected and presented in Table 6.7. Therefore the table shows the results of the proposed re-election criteria. The proposed method chooses controller A to replace controller B and vice versa, while choosing controller B to replace controller C.

Table 6.7. The local controllers with their replacements

Local Controller	Replacement	Distance (m)
A	B	97
B	A	97
C	B	268

Figure 6.9 below shows the results of the second experiment, the latency variations before the controller was failed and after. We performed all the failing on controller B, any controller could be used (failed) and that would not change the outcome of the experiment. According to the results in Table 6.7, the proposed method will always choose the closest controller. However, the current method could choose any of the remaining two. The results in Figure 6.9 below shows the latency before controller B is failed, then latency when the closest controller is chosen, and lastly latency when the furthest controller is chosen. The last two results are applicable to the current method because the choosing is random. In Figure 6.9, **B** refers to controller B before the failing, **B->A** refers to the replacement of controller B by controller A while **B->C** refers to the replacement of controller B by controller C.

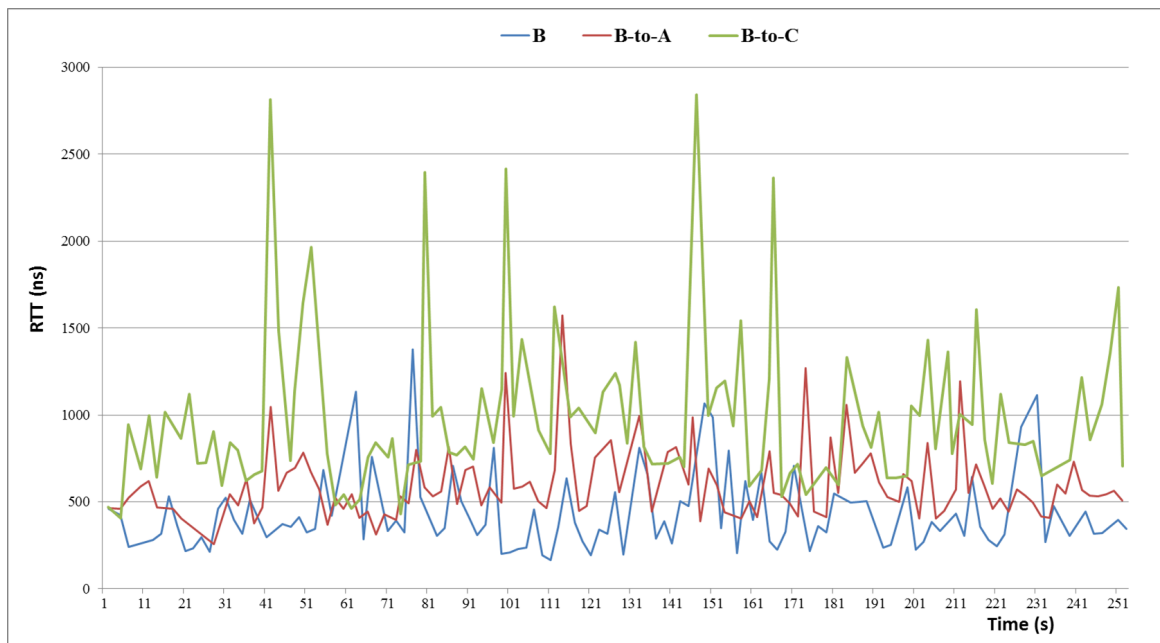


Figure 6.9. The latency of the two re-election criteria, the proposed and the current.

The latency results presented show a slight addition of delay as the change of controllers takes place. Accordingly, the results show that the added delay is more when the controller replacement does not take distance into account. This is of significant importance in SDWSN. This difference could be deemed insignificant in traditional SDN, but very critical in SDWSNs. Considering the distances between the controllers, the speed in traditional networks could overwrite the latency differences, but remains critical in SDWSNs.

As previously stated, specifically Section 5.2, the connection of the sink nodes to the controller is manual. As a result, when the controller is stopped, the simulation stops, which contrasts the ONOS which invokes the re-election service. Therefore, we look at the choice of ONOS in terms of which controller it chooses as a replacement. We then connect the sink(s) to that controller, then measure the latency again. The automatic does not change or negate the results because the primary purpose is to evaluate the latency with the different controller locations. As emphasised previously, this connection should be automatic so that it could correspond with the controller's redundancy plan. In contrast, in the traditional SDN, the controller self-discover the devices (switches).

6.3 CONCLUSION

This chapter serves as an enhancement of the proposed fragmentation model in terms of latency and reliability or resilience. It seeks to strengthen and optimise the fragmentation model for real-life operation. We discussed in detail two problems, the controller placement and the controller mastership re-election. The controller placement seeks to reduce the propagation latency between the sink nodes and the local controllers, as well as between the local controllers and the global controller. The experiments showed that we cannot use the same method of optimal placement for both the local controllers and the global controller. Therefore, two models of optimal controller placement were discussed and adopted. K-means for the local controller placement and re-optimised k-means or k-center for the global controller placement. We also enhanced the current ONOS mastership re-election criteria with distance consideration in line with the ideals of the fragmentation model. We adopted the Haversine formula and implemented it in ONOS to ensure that the re-election was in line with the fragmentation model. These two problems are intertwined and complementary. This ensures that propagation latency is kept to a minimum and at an acceptable threshold after a controller failure. The change of controller inevitably adds to the latency and the role of this exercise is to ensure that even after failure, the network does not suffer abnormal latencies to the detriment of the network.

CHAPTER 7 CONCLUSION

7.1 CONCLUSION

The software-defined wireless sensor network is a new and emerging network model that seeks to improve the efficiency of wireless sensor networks. It is formed by applying software-defined networking to WSN with the aim of addressing the WSN's inherent challenges which have been a major obstacle to their efficacy and applicability. WSN has always been riddled with constraints such as limited energy, bandwidth, memory, and processing. Over the years, several efforts at meeting these challenges have been ineffectual. The prevalence of SDN in recent years has gained much traction as systems seek to capitalise on the premise that it brings to computing and networking. SDN separates the control logic from the forwarding engine on the network elements, thereby leaving them as sheer skeletal devices that only understand the instructive language of their controller(s). The control intelligence is centralised in a controller which controls the entire network. Application of SDN in WSNs is similar; bringing much respite to the severely constrained sensor nodes. Hypothetically, this implies that most of the energy-intensive functions would reside in the controller instead of the device. This evolves to the SDWSN paradigm.

The SDWSN is envisaged to play a vital role in the Internet of Things ecosystem. The IoT is at the centre of the new wave of development called the 4th Industrial Revolution or 4IR. This defines a new era of digitisation where automation and data technologies are the kingpins of the new industrial development. Some of the technologies envisaged to stimulate this trend are the Industrial Internet of Things, artificial intelligence, virtual realities, cyber-physical systems, cloud, and cognitive computing. The SDN model is already being applied to most of these technologies. The SDWSN is at the base of most of these technologies and therefore very essential and central to the success of this evolution.

The major challenge facing the SDWSN model is found on the controller. The centralisation of the controller entails many benefits but, worrying disadvantages too. Some of those concern the reliability and security of the controller, given its enormous responsibilities. A central controller could be a central point of failure and a conspicuous target for adversaries. Any failure of the controller effectively affects the entire network as it holds the whole operational intelligence. Another challenge is the possibility that as the network grows, performance might be impacted. Also, as the sensor nodes move farther away from the controller, the distance can also affect the performance negatively. The distance can also affect the responsiveness of the controller and thereby render the network inefficient because the SDWSN carries live data which changes rapidly. In the delay-sensitive IIoT this could have dire effects on production etc.

Nevertheless, due to the infancy of the SDWSN model, most of the current literature employs a central controller. The challenge is therefore to distribute the control logic to offer reliability, performance, and scalability. Furthermore, the challenge is not only to distribute the control logic but also to find an efficient way through which this could be done without compromising any of the quality imperatives.

This study had two major objectives: to investigate if a distributed-control system is ideal for SDWSN and to investigate the plausibility of using fragmentation to distribute the SDWSN control logic. Fragmentation entails distributing the control logic to different segments (fragments), each responsible for a particular cluster. This method consists of small and lean local controllers closer to the infrastructure elements. These local controllers operate independently, and only occasionally with the global controller. The fragmentation model uses the eventual consistency data model, which consists of Best-effort and Anti-entropy algorithms. These algorithms are restructured and reused to ensure fragmentation.

The evaluations showed that indeed a single central controller is not ideal for the SDWSN network. Besides the hypotheses, the performance of the distributed controllers proved superior to that of the central controller. The results showed a huge disparity between the two architectures. The two distributed-control architectures improve the time delay by almost 50% compared to the central controller. This is a huge improvement that affirms the applicability of the distributed controllers. The performance comparison between the two distributed-control architectures showed an improvement on the part of the fragmentation model. The fragmentation model performed slightly better than the

original version used for benchmarking. Thus, the fragmentation model proved particularly efficient with less delay overall.

The main aim of this study was to achieve scalability, reliability, and performance. Regarding scalability, the fragmentation model performed well consistently as the number of nodes increased. The consistency in performance amid topology changes affirm the ability of the distributed architecture to handle dynamic changes in the network smoothly. The distributed controllers bring more stability to performance as the nodes share the load. In addition, the fragmentation model enables parallelism. It also ensures that there is redundancy. The reliability was evaluated in terms of partition tolerance, which the multiple controllers in the distributed architectures enable. Thus, the tolerance in node failure or unavailability provides cushion and assurance to various applicable systems. Therefore, we can conclude that the fragmentation model does improve the efficiency without compromising the quality imperatives. The fragmentation model is also shown not to negate the efficiency of the network.

The fragmentation model suffered an addition of a fraction of controller connection delay compared to the original version due to the two-level control architecture. However, this delay is still within the margins of the original version. This does not negate the performance as it only affects synchronisation at start-up. Evidence of this is the efficient subsequent performance due to the improved time complexity of the algorithms as the local controllers come into effect. The time complexity or the efficiency of the algorithms improved, and this is proven theoretically by the Big O notation and practically affirmed by the results. The efficiency in controller response time and efficient time complexity is vital as it averts issues such as throughput, congestion, and delay.

As the fragmentation model is ideally earmarked to play a critical role in stimulating the participation of SDWSN in the IoT ecosystem, we optimise the model for integration and operational efficiency. We consider two aspects: the controller placement and controller re-election after failure. The controller placement entrenches the fragmentation model as it ensures that there is optimal placement of the local controllers on the network. The main aim is to reduce the propagation latency between the resource-constrained sink nodes and the local controllers, as well as between the local controllers and the global controller. Two methods of controller placement which use the k-center optimisation technique are adopted and discussed in detail. As this strengthens the fragmentation model's ideal of placing the controllers closer to the infrastructure elements, the challenge arises when a controller failure occurs

because the current implementation chooses the replacement controller in a listed chronological order. This is an antithesis of the fragmentation model because the chosen replacement could be far from the cluster. This could result in excessive latency. Therefore, we enhanced the controller re-election criteria by adding a distance metric to the decision-making of the candidate replacement of a failed controller. This ensures that the system chooses a controller close to the failed controller or the cluster as a replacement amongst those available. This was evaluated in the ONOS system.

It can therefore be concluded that:

1. A single Central controller is not ideal for the software-defined wireless sensor networks' control.
2. Fragmentation can be used to distribute the SDWSN control system without compromising the quality imperatives.
3. Fragmentation can be used to distribute the SDWSN control system for optimal efficiency.
4. Fragmentation does bring reliability, scalability, and performance to the SDWSN control system.

7.1.1 SUMMARY OF CONTRIBUTIONS

The major contributions of the study can be summarised as follows:

1. A comprehensive survey of the current literature, challenges, and design requirements of the SDWSN. This study formed part of a published survey paper in a flagship journal which has already made and continues to make a huge impact in the research community. The essence of this survey study was to unravel all literature available in SDWSN to compare, contrast, and extract lessons that can be used for further research development, and to measure the current state of SDWSN and identify the challenges currently besetting it. This was the first comprehensive survey in this area. It delved into a variety of issues such as identifying the research gaps, issues pertaining to standardisation, and providing appropriate recommendations. This would greatly assist the research community. The paper also highlighted in much detail, the role of SDWSN in IoT and the future internet. This will surely spark an interest in the research community as well as the industry at large. The paper is already having a major impact as is evident from its read and citation statistics.

2. To design an architecture for SDWSN. An architecture is crucial for any developing project. It provides a reference, a guide, and basic requirements for SDWSN. Particularly important is the fact that it would create a commonality amongst different solutions which would ultimately result in compatible solutions which are critical in this age and are also in line with the principles of SDN. This architecture provides a holistic overview of SDWSN and its requirements. The design of this architecture is based on the current development, challenges, and future design requirements for SDWSN. This would contribute immensely to the IoT framework, in which the SDWSN is envisaged to play a huge role.
3. To establish that the central control architecture is not ideal for the SDWSN. This formed part of a published conference paper at a flagship conference. There are so many hypothetical works on central versus distributed controllers. This exercise was aimed at practically proving the hypothesis, especially from the SDWSN perspective. Although this has already been done on the traditional SDN with various distributed controllers, this was not the case for SDWSN, and given the unique characteristics of the SDWSN, it was imperative that this be carried out. This determination would assist other researchers and further strengthen the development of distributed-control solutions for SDWSN. As the SDWSN model grows and gains popularity and its role in the IoT strengthens, a distributed control is a necessity.
4. To establish that the fragmentation model can be used to distribute the control logic of the SDWSN. The findings of this are also contained in a journal paper published to a flagship high impact journal. There are different ways of distribution and the data set serves as a guide in choosing the appropriate distribution method. The scope and demand of sensory data have grown and have also been made complex by the imminent IoT service demands. A new method of handling this and the overall control of SDWSN is important. The determination that the fragmentation model can be used for this purpose is novel. This would have a great impact on how the SDWSN control systems are implemented in the future. This fragmentation takes into account the immense importance of the role of SDWSN in IoT, and the architecture is optimised for that role.
5. A model of distribution was designed, consisting of efficient algorithms, and it was successfully implemented; this is the Fragmentation model. Applications in the IoT framework and the future internet are envisioned to be mostly delay sensitive and thus require efficient services. The algorithms used in the implementation of the fragmentation model are efficient and improve the performance of the system. Efficient and high-performance control systems would result in efficient networks which would have a positive impact in IoT, industrial IoT, and stimulate the

4th industrial revolution. The fragmentation would enable the SDWSN to be scalable, efficient, reliable, and cost effective.

6. To propose an optimal controller placement method which supports the fragmentation model. This ensures that the local controllers are placed at the centres of their respective clusters or at an equal distance to the sink nodes. It also ensures that the latency between the sink nodes and the local controllers and between the local controllers and the global controller is minimised.
7. To propose a location-aware controller replacement criterion which supports the fragmentation model. This ensures that the controller replacement after failure is in line with the initial placement consideration fragmentation which places the local controller closer to the infrastructure elements. It also ensures that the latency accrued due to controller change is within the acceptable threshold.

7.2 FUTURE RESEARCH WORK

The SDWSN is still in developmental stages and therefore there is so much work to be done as highlighted in Chapter 2. One of the challenges identified is practical implementation of the SDWSN on real networks. This would allow the proposed distributed-control system to be further tested using real network testbeds. This model needs to be further tested for scalability in a large network. The model was implemented in ONOS 1.0.2 according to the SDN-WISE solution. This version uses Hazelcast which has been proven not to handle the split-brain situation well. The fragmentation model needs to be implemented in the latest versions of ONOS which now uses Atomix instead of Hazelcast. The fragmentation model could also be implemented in another distributed controller and or tested using other simulations available such as IT-SDN [225]. The fragmentation model should also be extended to check the viability of fragmenting different sensory traits; this would extend the practical evaluation highlighted above. Security is another grey area that needs urgent research consideration in SDWSN.

REFERENCES

- [1] 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, no. 2, pp. 1872–1899, 2017.
- [2] I. Howitt and J. Gutierrez, "IEEE 802.15.4 low rate - Wireless Personal Area Network Coexistence Issues," in *Proceedings of the IEEE Wireless Communications and Networking - WCNC 2003*, 2003, pp. 1481–1486.
- [3] W. Braun and M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014.
- [4] S. Shanmugapriya and M. Shivakumar, "Context Based Route Model for Policy Based routing in WSn using SDN approach," in *Proceedings of the BGSIT National Conference on Emerging Trends in Electronics and Communication*, 2015, pp. 1–8.
- [5] Z. Han and W. Ren, "A Novel Wireless Sensor Networks Structure Based on the SDN," *International Journal of Distributed Sensor Networks*, vol. 10, no. 3, pp. 1–8, 2014.
- [6] A. A. S. Yuan, H.-T. H. Fang, and Q. Wu, "OpenFlow based hybrid routing in Wireless Sensor Networks," in *Proceedings of the Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2014, pp. 1–5.
- [7] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. Mck-

- eown, "OpenRoads : Empowering Research in Mobile Networks," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 125–126, 2010.
- [8] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "Fragmentation-based Distributed Control System for Software Defined Wireless Sensor Networks," *IEEE Industrial Informatics*, vol. In Press, 2018.
- [9] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
- [10] Cisco Systems Inc, "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are," White Paper, San Jose, CA, USA, 2016. [Online]. Available: http://www.cisco.com/c/dam/en{_}us/solutions/trends/iot/docs/computing-overview.pdf (Accessed on: 2016-08-11).
- [11] P. Hu, "A System Architecture for Software-Defined Industrial Internet of Things," in *Proceedings of the IEEE International Conference on Ubiquitous Wireless Broadband - ICUWB 2015*, 2015, pp. 1–5.
- [12] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turetletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.
- [13] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
- [14] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and Haiyong Xie, "A Survey on Software-Defined Networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27 – 51, 2014.
- [15] N. A. Jagadeesan and B. Krishnamachari, "Software-Defined Networking Paradigms in Wireless Networks: A Survey," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, pp. 1–11, 2014.

- [16] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are We ready for SDN? Implementation challenges for Software-Defined Networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.
- [17] J. Li, L. Huang, Y. Zhou, S. He, and Z. Ming, "Computation Partitioning for Mobile Cloud Computing in a Big Data Environment," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 2009–2018, 2017.
- [18] D. A. Chekired, L. Khoukhi, and H. T. Mouftah, "Decentralized Cloud-SDN Architecture in Smart Grid: A Dynamic Pricing Model," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 3, pp. 1220 – 1231, 2018.
- [19] G. S. Aujla, N. Kumar, A. Y. Zomaya, and R. Rajan, "Optimal Decision Making for Big Data Processing at Edge-Cloud Environment: An SDN Perspective," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 2, pp. 778 – 789, 2018.
- [20] M. Jacobsson and C. Orfanidis, "Using Software-defined Networking Principles for Wireless Sensor Networks," in *Proceedings of the 11th Swedish National Computer Networking Workshop - SNCNW 2015*, 2015, pp. 1–5.
- [21] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless Sensor Network Survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.
- [22] G. Akpakwu, B. Silva, G. Hancke, and A. Abu-Mahfouz, "A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges," *IEEE Access*, vol. 6, no. 12, pp. 3619 – 3647, 2017.
- [23] K. M. Modieginyane, B. B. Letswamotse, R. Malekian, and A. M. Abu-Mahfouz, "Software Defined Wireless Sensor Networks Application Opportunities for efficient Network Management: A survey," *Computers & Electrical Engineering*, vol. 66, no. 2, pp. 274–287, 2017.
- [24] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, ser.

- USENIX ATC'14, 2014, pp. 305–320.
- [25] L. Lamport and Leslie, “The Part-time Parliament,” *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, 1998.
- [26] L. Lamport, “Paxos Made Simple,” *ACM SIGACT News*, vol. 32, no. 4, pp. 51–58, 2001.
- [27] A. De Gante, M. Aslan, and A. Matrawy, “Smart Wireless Sensor Network Management based on Software-Defined Networking,” in *Proceedings of the 27th Biennial Symposium on Communications (QBSC)*, 2014, pp. 71–75.
- [28] D. Kreutz, F. M. Ramos, and P. Verissimo, “Towards Secure and Dependable Software-Defined Networks,” in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, 2013, pp. 55–60.
- [29] S. T. Ali, V. Sivaraman, A. Radford, and S. Jha, “A Survey of Securing Networks Using Software Defined Networking,” *IEEE Transactions on Reliability*, vol. 64, no. 3, pp. 1086 – 1097, 2015.
- [30] J. Sherry and S. Ratnasamy, “A Survey of Enterprise Middlebox Deployments,” University of California, Tech. Rep. EECS-2012-24, 2012. [Online]. Available: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-24.pdf> (Accessed on: 2016-08-08).
- [31] I. Kovacevic, “FoRCES Protocol as a Solution for Interaction of Control and Forwarding Planes in Distributed Routers,” in *Proceedings of the 17th Telecommunications forum TELFOR*, 2009, pp. 529–532.
- [32] E. Haleplidis, J. Hadi Salim, J. M. Halpern, S. Hares, K. Pentikousis, K. Ogawa, W. Weiming, S. Denazis, and O. Koufopavlou, “Network Programmability With ForCES,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1423–1440, 2015.

- [33] F. Hu, Q. Hao, and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.
- [34] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using OpenFlow: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493 – 512, 2013.
- [35] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," White Paper, CA, USA, 2012. [Online]. Available: http://www.bigswitch.com/sites/default/files/sdn{_}resources/onf-whitepaper.pdf (Accessed on: 2015-09-18).
- [36] E. Haleplidis, S. Denazis, O. Koufopavlou, J. H. Salim, and J. Halpern, "Software-Defined Networking: Experimenting with the Control to Forwarding Plane Interface," in *Proceedings of the European Workshop on Software Defined Networks, EWSDN 2012*, 2012, pp. 91–96.
- [37] J. Qadir, N. Ahmed, and N. Ahad, "Building Programmable Wireless Networks: An Architectural Survey," *EURASIP JWCN*, vol. 2014, no. 1, pp. 1–19, 2014.
- [38] D. Zeng, T. Miyazaki, S. Guo, T. Tsukahara, J. Kitamichi, and T. Hayashi, "Evolution of Software-Defined Sensor Networks," in *Proceedings of the IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*, 2013, pp. 410–413.
- [39] M. P. Fernandez, "Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive," in *Proceedings of the International Conference on Advanced Information Networking and Applications, AINA*, 2013, pp. 1009–1016.
- [40] B. Heller, "OpenFlow Switch Specification," 2016-10-18, Tech. Rep. ONF TS-012, 2012. [Online]. Available: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf> (Accessed on: 2016-05-27).
- [41] Y. Jarraya, T. Madi, and M. Debbabi, "A Survey and a Layered Taxonomy of Software-Defined Networking," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1955–1980, 2014.

- [42] S. Jain, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, and J. Zhou, “B4: Experience with a Globally-Deployed Software Defined WAN,” in *Proceedings of the ACM SIGCOMM conference - SIGCOMM '13*, 2013, pp. 3–14.
- [43] M. Bansal, J. Mehlman, S. Katti, and P. Levis, “OpenRadio: A Programmable Wireless Data-plane,” in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 109–114.
- [44] L. Suresh and J. Schulz-Zander, “Towards Programmable Enterprise WLANs with Odin,” in *Proceedings of the first Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 115–120.
- [45] Y. Jararweh, M. A. Ayyoub, A. Doulat, A. A. A. A. Aziz, H. A. B. Salameh, and A. A. Khreishah, “SD-CRN: Software Defined Cognitive Radio Network Framework,” in *Proceedings of the IEEE International Conference on Cloud Engineering*, 2014, pp. 592–597.
- [46] G. Sun, G. Liu, and Y. Wang, “SDN architecture for Cognitive Radio Networks,” in *Proceedings of the 1st International Workshop on Cognitive Cellular Systems (CCS)*, 2014, pp. 1–5.
- [47] V. Nascimento, M. Moraes, R. Gomes, and B. Pinheiro, “Filling the Gap Between Software Defined Networking and Wireless Mesh Networks,” in *Proceedings of the 10th International Conference on Network and Service Management (CNSM) and Workshop*, 2014, pp. 451–454.
- [48] P. Dely, A. Kassler, and N. Bayer, “OpenFlow for Wireless Mesh Networks,” in *Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, 2011, pp. 1–6.
- [49] N. V. R. Gupta and M. V. Ramakrishna, “A Road Map for SDN-Open Flow Networks,” *International Journal of Modern Communication Technologies & Research (IJMCTR)*, vol. 3, no. 6, pp. 38–46, 2015.
- [50] K.-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, and N. McKeown, “The

- Stanford OpenRoads deployment,” in *Proceedings of the 4th ACM international workshop on Experimental evaluation and characterization - WINTECH '09*, 2009, pp. 59–66.
- [51] A. Mahmud and R. Rahmani, “Exploitation of OpenFlow in Wireless Sensor Networks,” in *Proceedings of the International Conference on Computer Science and Network Technology (Iccsnt)*, 2012, pp. 594–600.
- [52] I. T. Haque and N. Abu-Ghazaleh, “Wireless Software Defined Networking: a Survey and Taxonomy,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2713–2737, 2016.
- [53] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, “Security in Software Defined Networks: A Survey,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2317 – 2346, 2015.
- [54] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, “Model Checking Invariant Security Properties in OpenFlow,” in *Proceedings of the IEEE International Conference on Communications*, 2013, pp. 1974–1979.
- [55] S. Shin, P. Porras, V. Yegneswaran, and M. Fong, “FRESCO: Modular Composable Security Services for Software-Defined Networks,” in *Proceedings of the ISOC Network and Distributed System Security Symposium*, 2013, pp. 1–16.
- [56] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “NOX: Towards an Operating System for Networks,” vol. 38, no. 3, pp. 105–110, 2008.
- [57] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, “A Security Enforcement Kernel for OpenFlow Networks,” in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12*, 2012, pp. 121–126.
- [58] 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, no. 10, pp. 2055–2064, 2015.

- [59] R. Beckett, X. K. Zou, S. Zhang, S. Malik, J. Rexford, and D. Walker, "An Assertion Language for Debugging SDN Applications," in *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14*, 2014, pp. 91–96.
- [60] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying Network-Wide Invariants in Real Time," in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12*, 2012, pp. 49–54.
- [61] E. Al-Shaer and S. Al-Haj, "FlowChecker: Configuration Analysis and Verification of Federated Openflow Infrastructures," in *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration - SafeConfig '10*, 2010, pp. 37–44.
- [62] P. Baronti, P. Pillai, V. W. Chook, S. Chessa, A. Gotta, and Y. F. Hu, "Wireless Sensor Networks: A Survey on the State of the Art and the 802.15.4 and Zigbee Standards," *Computer Communications*, vol. 30, no. 7, pp. 1655–1695, 2007.
- [63] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy Conservation in Wireless Sensor Networks: A Survey," *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, 2009.
- [64] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless Sensor Networks : A Survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.
- [65] Y.-F. Lee and C.-C. Shen, "A Transaction-based Approach to Over-the-air Programming in Wireless Sensor Networks," in *Proceedings of the International Symposium on Communications and Information Technologies*, 2007, pp. 1377–1382.
- [66] J. Hughes, J. Yan, and K. Soga, "Development of Wireless Sensor Network Using Bluetooth Low Energy (BLE) for Construction Noise Monitoring," *International Journal on Smart Sensing and Intelligent Systems*, vol. 8, no. 2, pp. 1379–1405, 2015.
- [67] IEEE Standard, "IEEE Standard Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs),"

- IEEE Standard, 2006. [Online]. Available: <http://ieeexplore.ieee.org/ielD/4152702/4152703/04152704.pdf> (Accessed on: 2017-02-16).
- [68] T. Luo, H.-P. Tan, and T. Q. S. Quek, "Sensor OpenFlow: Enabling software-defined wireless sensor networks," *IEEE Communications Letters*, vol. 16, no. 11, pp. 1896–1899, 2012.
- [69] S. Petersen and S. Carlsen, "WirelessHART Versus ISA100.11a: The Format War Hits the Factory Floor," *IEEE Industrial Electronics Magazine*, vol. 5, no. 4, pp. 23–34, 2011.
- [70] T. Lennvall, S. Svensson, and F. Hekland, "A comparison of WirelessHART and ZigBee for industrial applications," in *Proceedings of the IEEE International Workshop on Factory Communication Systems, WFCS*, 2008, pp. 85–88.
- [71] Y. Mazzer and B. Tourancheau, "Comparisons of 6LoWPAN Implementations on Wireless Sensor Networks," in *Proceedings of the 3rd International Conference on Sensor Technologies and Applications*, 2009, pp. 689–692.
- [72] K. Akkaya and M. Younis, "A survey on Routing Protocols for Wireless Sensor Networks," *Ad hoc networks*, vol. 3, no. 3, pp. 325–349, 2005.
- [73] M. J. Mudumbe and A. M. Abu-Mahfouz, "Smart Water Meter System for User-centric Consumption Measurement," in *Proceedings of the IEEE 13th International Conference on Industrial Informatics (INDIN)*, 2015, pp. 993–998.
- [74] D. Christin, A. Reinhardt, P. S. Mogre, and R. Steinmetz, "Wireless Sensor Networks and the Internet of Things : Selected Challenges," in *Proceedings of the 8th GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze"*, 2009, pp. 31–34.
- [75] H. Modares, R. Salleh, and A. Moravejosharieh, "Overview of Security Issues in Wireless Sensor Networks," in *Proceedings of the third International Conference on Computational Intelligence, Modelling & Simulation*, 2011, pp. 308–311.

- [76] N. Ntuli and A. M. Abu-Mahfouz, "A Simple Security Architecture for Smart Water Management System," *Procedia Computer Science*, vol. 83, no. 04, pp. 1164–1169, 2016.
- [77] A. Pathan, Hyung-Woo Lee, and Choong Seon Hong, "Security in Wwireless Sensor Networks: Issues and Challenges," in *Proceedings of the 8th International Conference Advanced Communication Technology*, 2006, pp. 6–11.
- [78] K. CHELLI, "Security Issues in Wireless Sensor Networks: Attacks and Countermeasures," in *Proceedings of the World Congress on Engineering*, 2015, pp. 1–6.
- [79] V. Kumar, A. Jain, and P. N. Barwal, "Wireless Sensor Networks: Security Issues , Challenges and Solutions," *International Journal of Information & Computation Technology*, vol. 4, no. 8, pp. 859–868, 2014.
- [80] T. Zia and A. Zomaya, "Security Issues in Wireless Sensor Networks," in *Proceedings of the International Conference on Systems and Networks Communications (ICSNC'06)*, 2006, pp. 37–40.
- [81] J. Louw, G. Niezen, T. D. Ramotsoela, and A. M. Abu-Mahfouz, "A key Distribution Scheme using Elliptic curve Cryptography in Wireless Sensor Networks," in *Proceedings of the IEEE 14th International Conference on Industrial Informatics (INDIN)*, 2016, pp. 1166–1170.
- [82] C. Alcaraz, P. Najera, J. Lopez, and R. Roman, "Wireless Sensor Networks and the Internet of Things : Do We Need a Complete Integration ?" in *Proceedings of the 1st International Workshop on the Security of the Internet of Things (SecIoT'10)*, 2010, pp. 1–8.
- [83] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software Defined Wireless Networks: Unbridling SDNs," in *Proceedings of the European Workshop on Software Defined Networks, EWSDN 2012*, 2012, pp. 1–6.
- [84] P. Jayashree and F. Infant Princy, "Leveraging SDN to conserve energy in WSN-An analysis," in *Proceedings of the 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*, 2015, pp. 1–6.

- [85] Y. Wang, H. Chen, X. Wu, and L. Shu, "An energy-efficient SDN based sleep scheduling algorithm for WSNs," *Journal of Network and Computer Applications*, vol. 59, no. 1, pp. 39–45, 2015.
- [86] B. T. de Oliveira, C. B. Margi, and L. B. Gabriel, "TinySDN: Enabling Multiple Controllers for Software-Defined Wireless Sensor Networks," in *Proceedings of the IEEE Latin-America Conference on Communications (LATINCOM)*, 2014, pp. 1–6.
- [87] P. Berde, W. Snow, G. Parulkar, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, and P. Radoslavov, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14*, 2014, pp. 1–6.
- [88] 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 - INM/WREN'10*, 2010, pp. 1–6.
- [89] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 351–362, 2011.
- [90] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, 2011.
- [91] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136 – 141, 2013.
- [92] K. Phemius, M. Bouet, and J. Leguay, "DISCO: Distributed multi-domain SDN controllers," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–4.
- [93] 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 - HotSDN '14*, 2014, pp. 133–138.
- [94] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, “Towards an Elastic Distributed SDN Controller,” *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 7–12, 2013.
- [95] A. M. Abu-Mahfouz and G. P. Hancke, “An Efficient Distributed Localisation Algorithm for Wireless Sensor Networks : based on Smart Reference-selection Method,” *International Journal of Sensor Networks*, vol. 13, no. 2, pp. 94–111, 2013.
- [96] Z. Shu, J. Wan, D. Li, J. Lin, A. V. Vasilakos, and M. Imran, “Security in Software-Defined Networking: Threats and Countermeasures,” *Mobile Networks and Applications*, vol. 21, no. 5, pp. 764–776, 2016.
- [97] I. Alsmadi and D. Xu, “Security of Software Defined Networks: A survey,” *Computers & Security*, vol. 53, no. 8, pp. 79–108, 2015.
- [98] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, “A survey of Mobile Cloud Computing: architecture, applications, and approaches,” *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013.
- [99] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, “Meridian: An SDN Platform for Cloud Network Services,” *IEEE Communications Magazine*, vol. 51, no. 2, pp. 120–127, 2013.
- [100] S. Azodolmolky, P. Wieder, and R. Yahyapour, “Cloud Computing Networking: Challenges and Opportunities for Innovations,” *IEEE Communications Magazine*, vol. 51, no. 7, pp. 54–62, 2013.
- [101] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, and A. Rindos, “SDIoT: A Software Defined based Internet of Things framework,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 6, no. 4, pp. 453–461, 2015.

- [102] Á. L. Valdivieso Caraguay, A. Benito Peral, L. I. Barona López, and L. J. García Villalba, “SDN: Evolution and Opportunities in the Development IoT Applications,” *International Journal of Distributed Sensor Networks*, vol. 10, no. 5, pp. 1896–1899, 2014.
- [103] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, “A Software Defined Networking architecture for the Internet-of-Things,” in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World*, 2014, pp. 1–9.
- [104] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog Computing and Its Role in the Internet of Things,” in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12*, 2012, pp. 13–16.
- [105] S. Yi, C. Li, and Q. Li, “A Survey of Fog Computing,” in *Proceedings of the 2015 Workshop on Mobile Big Data - Mobidata '15*, 2015, pp. 37–42.
- [106] I. Stojmenovic, “Fog Computing: A Cloud to the Ground Support for Smart Things and Machine-to-Machine Networks,” in *Proceedings of the Australasian Telecommunication Networks and Applications Conference (ATNAC)*, 2014, pp. 117–122.
- [107] N. B. Truong, G. M. Lee, and Y. Ghamri-Doudane, “Software Defined Networking-based Vehicular Adhoc Network with Fog Computing,” in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 1202–1207.
- [108] M. T. Beck, M. Werner, S. Feld, and T. Schimper, “Mobile Edge Computing : A Taxonomy,” in *Proceedings of the Sixth International Conference on Advances in Future Internet*, 2014, pp. 48–54.
- [109] I. Ku, Y. Lu, and M. Gerla, “Software-Defined Mobile Cloud: Architecture, services and use cases,” in *Proceedings of the International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2014, pp. 1–6.
- [110] N. Bhushan, J. Junyi Li, D. Malladi, R. Gilmore, D. Brenner, A. Damnjanovic, R. Sukhavasi,

- C. Patel, and S. Geirhofer, "Network Densification: The Dominant Theme for Wireless Evolution in 5G," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 82–89, 2014.
- [111] X. Ge, S. Tu, G. Mao, C.-X. Wang, and T. Han, "5G Ultra-Dense Cellular Networks," *IEEE Wireless Communications*, vol. 23, no. 1, pp. 72–79, 2016.
- [112] A. Osseiran, F. Boccardi, V. Braun, K. Kusume, P. Marsch, M. Maternia, O. Queseth, M. Schellmann, H. Schotten, H. Taoka, H. Tullberg, M. A. Uusitalo, B. Timus, and M. Fallgren, "Scenarios for 5G Mobile and Wireless Communications: The Vision of the METIS project," *IEEE Communications Magazine*, vol. 52, no. 5, pp. 26–35, 2014.
- [113] B. Soret, K. Pedersen, N. K. Jørgensen, and V. Fernández-López, "Interference Coordination for Dense Wireless Networks," *IEEE Communications Magazine*, vol. 53, no. 1, pp. 102–109, 2015.
- [114] T. S. Rappaport, S. Shu Sun, R. Mayzus, H. Hang Zhao, Y. Azar, K. Wang, G. N. Wong, J. K. Schulz, M. Samimi, and F. Gutierrez, "Millimeter Wave Mobile Communications for 5G Cellular: It Will Work!" *IEEE Access*, vol. 1, no. 5, pp. 335–349, 2013.
- [115] H. Ali-Ahmad, C. Cicconetti, A. de la Oliva, V. Mancuso, M. R. Sama, P. Seite, and S. Shanmugalingam, "An SDN-Based Network Architecture for Extremely Dense Wireless Networks," in *Proceedings of the IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013, pp. 1–7.
- [116] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and D. Soldani, "SDN-based 5G Mobile Networks: architecture, functions, procedures and backward compatibility," *Transactions on Emerging Telecommunications Technologies*, vol. 26, no. 1, pp. 82–92, 2015.
- [117] P. Rost, C. Bernardos, A. Domenico, M. Girolamo, M. Lalam, A. Maeder, D. Sabella, and D. Wübben, "Cloud Technologies for Flexible 5G Radio Access Networks," *IEEE Communications Magazine*, vol. 52, no. 5, pp. 68–76, 2014.
- [118] N. Omnes, M. Bouillon, G. Fromentoux, and O. Grand, "A programmable and virtualized network & IT infrastructure for the internet of things: How can NFV & SDN help for facing the

- upcoming challenges,” in *Proceedings of the 18th International Conference on Intelligence in Next Generation Networks*, 2015, pp. 64–69.
- [119] A.-C. G. Anadiotis, L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, “Towards a Software-Defined Network Operating System for the IoT,” in *Proceedings of the IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, pp. 579–584.
- [120] A. M. Abu-Mahfouz, T. Olwal, A. Kurien, J. L. Munda, and K. Djouani, “Toward developing a distributed autonomous energy management system (DAEMS),” in *Proceedings of the IEEE AFRICON 2015 Conference on Green Innovation for African Renaissance*, 2015, pp. 1–6.
- [121] A. M. Abu-Mahfouz, Y. Hamam, P. R. Page, K. Djouani, and A. Kurien, “Real-time Dynamic Hydraulic Model for Potable Water Loss Reduction,” *Procedia Engineering*, vol. 154, no. 07, pp. 99–106, 2016.
- [122] L. B. Campos, C. E. Cugnasca, A. R. Hirakawa, and J. S. C. Martini, “Towards an IoT-based system for Smart City,” in *Proceedings of the IEEE International Symposium on Consumer Electronics (ISCE)*, 2016, pp. 129–130.
- [123] S. Bhushan, B. Bohara, P. Kumar, and V. Sharma, “A new approach towards IoT by using health care-IoT and food distribution IoT,” in *Proceedings of the 2nd International Conference on Advances in Computing, Communication, & Automation (ICACCA) (Fall)*, 2016, pp. 1–7.
- [124] A. Luckshetty, S. Dontal, S. Tangade, and S. S. Manvi, “A survey: Comparative study of applications, attacks, security and privacy in VANETs,” in *Proceedings of the International Conference on Communication and Signal Processing (ICCSP)*, 2016, pp. 1594–1598.
- [125] S. Sadiq, S. Fizza, and M. Tahira, “A Survey on Wireless Software Defined Networks,” *International Journal of Computer and Communication System Engineering (IJCCSE)*, vol. 02, no. 01, pp. 155–159, 2015.
- [126] A. Malik, J. Qadir, B. Ahmad, K.-L. Alvin Yau, and U. Ullah, “QoS in IEEE 802.11-based Wireless Networks: A Contemporary Review,” *Journal of Network and Computer Applications*,

- vol. 55, no. 8, pp. 24–46, 2015.
- [127] A. Drescher, “A Survey of Software-Defined Wireless Networks,” 2014. [Online]. Available: <http://www.cse.wustl.edu/{~}jain/cse574-14/index.html> (Accessed on: 2014-04-30).
- [128] Q. Duan, Y. Yan, and A. V. Vasilakos, “A Survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing,” *IEEE Transactions on Network and Service Management*, vol. 9, no. 4, pp. 373–392, 2012.
- [129] J. Pan, S. Paul, and R. Jain, “A Survey of the Research on Future Internet Architectures,” *IEEE Communications Magazine*, vol. 49, no. 7, pp. 26–36, 2011.
- [130] M. Yang, Y. Li, D. Jin, L. Zeng, X. Wu, and A. V. Vasilakos, “Software-Defined and Virtualized Future Mobile and Wireless Networks: A Survey,” *Mobile Networks and Applications*, vol. 20, no. 1, pp. 4–18, 2015.
- [131] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.
- [132] L. M. Vaquero and L. Rodero-Merino, “Finding your Way in the Fog,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.
- [133] B. Pediredla, K. I. K. Wang, Z. Salcic, and A. Ivoghlian, “A 6LoWPAN Implementation for Memory Constrained and Power efficient Wireless Sensor Nodes,” in *Proceedings of the 39th Annual Conference of the IEEE Industrial Electronics Society - IECON 2013*, 2013, pp. 4432–4437.
- [134] C. P. P. Schumacher, N. Kushalnagar, and G. Montenegro, “IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals,” Internet Draft, Internet Engineering Task Force (IETF), 2007. [Online]. Available: <https://tools.ietf.org/html/rfc4919> (Accessed on: 2016-01-20).

- [135] S. A. Catapang, Z. J. M. Roberts, K. I.-K. Wang, and Z. Salcic, "An Infrastructure for Integrating Heterogeneous Embedded 6LoWPAN Networks for Internet of Things Applications," in *Proceedings of the Seventh International Conference on Sensing Technology (ICST)*, 2013, pp. 741–746.
- [136] 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 *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 513–521.
- [137] H. Huang, J. Zhu, and L. Zhang, "An SDN _ based Management Framework for IoT Devices," in *Proceedings of the Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CICT 2014). 25th IET*, 2014, pp. 175 – 179.
- [138] Z. Cai, A. L. Cox, and T. S. Eugene Ng, "Maestro: A System for Scalable OpenFlow Control," Rice University, Houston, TX, USA, Tech. Rep. TR10-11, 2010. [Online]. Available: <http://www.cs.rice.edu/~eugeneng/papers/TR10-11.pdf> (Accessed on: 2014-03-14).
- [139] D. Erickson, "The Beacon Openflow Controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013, pp. 13–18.
- [140] Project Floodlight, "Floodlight OpenFlow Controller - Project Floodlight." [Online]. Available: <http://www.projectfloodlight.org/floodlight/> (Accessed on: 2015-11-24).
- [141] S. H. Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12*, 2012, pp. 19–24.
- [142] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "ElastiCon: An Elastic Distributed SDN Controller," in *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems - ANCS '14*, 2014, pp. 17–28.
- [143] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on Network Virtualization Hyper-

- visors for Software Defined Networking,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 655 – 685, 2015.
- [144] F. Olivier, G. Carlos, and N. Florent, “SDN Based Architecture for Clustered WSN,” in *Proceedings of the 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2015, pp. 342–347.
- [145] P. Lin, J. Bi, Z. Chen, Y. Wang, H. Hu, and A. Xu, “WE-bridge: West-east bridge for SDN inter-domain network peering,” in *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2014, pp. 111–112.
- [146] A. Sehgal, “Using the contiki cooja simulator,” 2013. [Online]. Available: <http://cnds.eecs.jacobs-university.de/courses/iotlab-2013/cooja.pdf> (Accessed on: 2017-08-10).
- [147] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, “System Architecture Directions for Networked Sensors,” *ACM Sigplan Notices*, vol. 35, no. 11, pp. 93–104, 2000.
- [148] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, *TinyOS: An Operating System for Wireless Sensor Networks*, W. Weber, J. Rabaey, and E. Aarts, Eds., 2005.
- [149] J. Qadir and O. Hasan, “Applying Formal Methods to Networking: Theory, Techniques, and Applications,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 256–291, 2015.
- [150] B. Lantz, B. Heller, and N. McKeown, “A Network in a Laptop,” in *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*, 2010, pp. 1–6.
- [151] S. Y. Shie-Yuan Wang, C. L. Chih-Liang Chou, and C. M. Chun-Ming Yang, “EstiNet OpenFlow Network Simulator and Emulator,” *IEEE Communications Magazine*, vol. 51, no. 9, pp. 110–117, 2013.
- [152] POX, “POX controller.” [Online]. Available: <http://www.noxrepo.org/pox/about-pox/> (Accessed on: 2016-06-09).

- [153] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya, "CloudSimSDN: Modeling and Simulation of Software-Defined Cloud Data Centers," in *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015, pp. 475–484.
- [154] T. R. Henderson, M. Lacage, and G. F. Riley, "Network Simulations with the ns-3 Simulator (2008)," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, 2008, pp. 1–1.
- [155] TREMA, "Trema." [Online]. Available: <https://trema.github.io/trema/> (Accessed on: 2016-06-09).
- [156] M. B. Al-Somaidai and E. B. Yahya, "Survey of Software Components to Emulate Open-Flow Protocol as an SDN Implementation," *American Journal of Software Engineering and Applications*, vol. 3, no. 6, pp. 74–82, 2014.
- [157] A. Mahmud, R. Rahmani, and T. Kanter, "Deployment of Flow-Sensors in Internet of Things' Virtualization via OpenFlow," in *Proceedings of the Third FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing*, 2012, pp. 195–200.
- [158] A. Hakiri, P. Berthou, A. Gokhale, and S. Abdellatif, "Publish/subscribe-enabled Software Defined Networking for Efficient and Scalable IoT Communications," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 48–54, 2015.
- [159] R. Huang, X. Chu, J. Zhang, and Y. H. Hu, "Energy-Efficient Monitoring in Software Defined Wireless Sensor Networks Using Reinforcement Learning : A Prototype," *International Journal of Distributed Sensor Networks*, vol. 11, no. 10, pp. 360–428, 2015.
- [160] R. Sherwood, G. Gibb, K.-k. Yap, G. Appenzeller, M. Casado, N. Mckeown, and G. Parulkar, "FlowVisor: A Network Virtualization Layer," Tech. Rep. OPENFLOW-TR-2009-1, 2009, (Accessed on: 2015-09-29).
- [161] R. Syyed, S. Kundu, C. Warty, and S. Nema, "Resource Optimization Using Software Defined

- Networking for Smart Grid Wireless Sensor Network,” in *Proceedings of the 3rd International Conference on Eco-friendly Computing and Communication Systems*, 2014, pp. 200–205.
- [162] C. Perkins, E. Belding-Royer, and S. Das, “RFC 3561-Ad hoc On-demand Distance Vector (AODV) routing,” *Internet RFCs*, pp. 1–37, 2003. [Online]. Available: <http://www.rfc-editor.org/info/rfc3561> (Accessed on: 2017-01-25).
- [163] OpenVswitch, “Open vSwitch.” [Online]. Available: <http://openvswitch.org/> (Accessed on: 2015-11-24).
- [164] H. Kim and N. Feamster, “Improving Network Management with Software Defined Networking,” *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.
- [165] A. Voellmy, H. Kim, and N. Feamster, “Procera: A Language for High-Level Reactive Network Control,” in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12*, 2012, pp. 43–48.
- [166] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, “Frenetic: A Network Programming Language,” *ACM SIGPLAN Notices*, vol. 46, no. 9, pp. 279–291, 2011.
- [167] A. Voellmy and P. Hudak, “Nettle: Taking the Sting Out of Programming Network Routers,” in *Proceedings of the 13th international conference on Practical aspects of declarative languages*, 2011, pp. 235–249.
- [168] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, “Composing Software-Defined Networks,” in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation*, 2013, pp. 1–13.
- [169] T. Hinrichs, N. Gude, M. Casado, J. Mitchell, and S. Shenker, “Practical Declarative Network Management,” in *Proceedings of the 1st ACM workshop on Research on Enterprise Networking*, 2009, pp. 1–10.

- [170] K. Feng, X. Huang, and Z. Su, "A Network Management Architecture for 6LoWPAN Network," in *Proceedings of the 4th IEEE International Conference on Broadband Network and Multimedia Technology, IC-BNMT 2011*, 2011, pp. 430–434.
- [171] H. Mukhtar, K. Kang-myoo, S. A. Chaudhry, A. H. Akbar, K. Ki-Hyung, and S.-W. Yoo, "LNMP- Management Architecture for IPv6 based low- power Wireless Personal Area Networks (6LoWPAN)," in *Proceedings of the IEEE Network Operations and Management Symposium - NOMS 2008*, 2008, pp. 417–424.
- [172] H. Choi, N. Kim, and H. Cha, "6LoWPAN-SNMP: Simple Network Management Protocol for 6LoWPAN," in *Proceedings of the 11th IEEE International Conference on High Performance Computing and Communications*, 2009, pp. 305–313.
- [173] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Rules Placement Problem in OpenFlow Networks: a Survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1273 – 1286, 2015.
- [174] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, Applications and Research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.
- [175] J. Saraswat and P. P. Bhattacharya, "Effect of Duty Cycle on Energy," *International Journal of Computer Networks & Communications (IJCNC)*, vol. 5, no. 1, pp. 125–140, 2013.
- [176] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "A Replication Component for Resilient OpenFlow-based Networking," in *Proceedings of the IEEE Network Operations and Management Symposium*, 2012, pp. 933–939.
- [177] T. Tsou, P. Aranda, H. Xie, R. Sidi, H. Yin, and D. Lopez, "SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains," Internet Draft, Internet Engineering Task Force (IETF), 2012. [Online]. Available: <https://tools.ietf.org/html/draft-yin-sdn-sdni-00> (Accessed on: 2016-01-20).
- [178] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievskiy, M. Zhuy, R. Ramanathany,

- Y. Iwataz, H. Inouez, T. Hamaz, S. Shenkerx, and K. et al., "Onix: A Distributed Control Platform for Large-scale Production Networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, USENIX Association, 2010, pp. 351–364.
- [179] M. Sneps-Sneppe and D. Namiot, "Metadata in SDN API for WSN," in *Proceedings of the 7th International Conference on New Technologies, Mobility and Security (NTMS)*, 2015, pp. 1–5.
- [180] X. Feng, J. Shen, and Y. Fan, "REST: An Alternative to RPC for Web Services Architecture," in *Proceedings of the First International Conference on Future Information Networks*, 2009, pp. 7–10.
- [181] R. Perrey and M. Lycett, "Service-Oriented Architecture," in *Proceedings of the Symposium on Applications and the Internet Workshops*, 2003, pp. 116–119.
- [182] R. Wallner and R. Cannistra, "An SDN Approach: Quality of Service using Big Switch's Floodlight Open-source Controller," in *Proceedings of the Asia-Pacific Advanced Network*, 2013, pp. 14–19.
- [183] RYU, "Ryu SDN Framework." [Online]. Available: <https://osrg.github.io/ryu/> (Accessed on:2017-01-10).
- [184] J. Stribling, Y. Sovran, I. Zhang, X. Pretzer, J. Li, M. F. Kaashoek, and R. Morris, "Flexible, Wide-Area Storage for Distributed Systems with WheelFS," in *Proceedings of the 6th USENIX symposium on Networked systems design and implementation - NSDI'09*, 2009, pp. 43–58.
- [185] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," in *Proceedings of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014, pp. 1–6.
- [186] Z. K. Khattak, M. Awais, and A. Iqbal, "Performance evaluation of OpenDaylight SDN controller," in *Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2014, pp. 671–676.

- [187] B. T. de Oliveira and C. B. Margi, "Distributed Control Plane Architecture for Software-defined Wireless Sensor Networks," in *Proceedings of the IEEE International Symposium on Consumer Electronics (ISCE)*, 2016, pp. 85–86.
- [188] SDN-WISE, "Controlling Heterogeneous Networks Using SDN-WISE and ONOS." [Online]. Available: <http://sdn-wise.dieei.unict.it/docs/guides/GetStartedONOS.html> (Accessed on: 2017-04-26).
- [189] H. I. Kobo, G. P. Hancke, and A. M. Abu-Mahfouz, "Towards A Distributed Control System For Software Defined Wireless Sensor Networks," in *Proceedings of the 43rd IEEE conference of Industrial Electronic Society - IECON 2017*, 2017, pp. 6125–6130.
- [190] P. Bailis and A. Ghodsi, "Eventual Consistency Today: Limitations, Extensions, and Beyond," *Queue*, vol. 11, no. 3, pp. 20–32, 2013.
- [191] C. Coronel and S. Morris, *Database Systems: Design Implementation and Management 11e*, 11th ed., 2015. [Online]. Available: www.cengage.com
- [192] S. I. Khan and A. S. M. L. Hoque, "A New Technique for Database Fragmentation in Distributed Systems," *International Journal of Computer Applications*, vol. 5, no. 9, pp. 20–24, 2010.
- [193] V. Ciriani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Fragmentation Design for Efficient Query Execution over Sensitive Distributed Databases," in *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems*, 2009, pp. 32–39.
- [194] J. Holliday, R. Steinke, D. Agrawal, and A. El Abbadi, "Epidemic Algorithms for Replicated Databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 5, pp. 1218–1238, 2003.
- [195] D. Shah, "Gossip Algorithms," *Foundations and Trends® in Networking*, vol. 3, no. 1, pp. 1–125, 2007.

- [196] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié, “Epidemic Information Dissemination in Distributed Systems,” *Computer*, vol. 37, no. 5, pp. 60–67, 2004.
- [197] A. Montresor, M. Jelasity, and O. Babaoglu, “Robust Aggregation Protocols for Large-Scale Overlay Networks,” in *Proceedings of the International Conference on Dependable Systems and Networks, 2004*, 2004, pp. 19–28.
- [198] S. Voulgaris and M. van Steen, “An Epidemic Protocol for Managing Routing Tables in Very Large Peer-to-Peer Networks,” in *Proceedings of the International Workshop on Distributed Systems: Operations and Management*, 2003, pp. 41–54.
- [199] B. Margi, “Building internet of things over software defined wireless sensor networks,” presented at Grace Hopper Celebration of Women in Computing (GHC), PALO ALTO, CA, USA, 2014.
- [200] A. Montresor, “Gossip protocols for large-scale distributed systems,” presented at the 28th Brazilian Symposium on Computer Networks and Distributed Systems, Gramado, Brazil, pp. 6125–6130, 2010.
- [201] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, “The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations,” in *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, 2004, pp. 79–98.
- [202] M. Jelasity, A. Montresor, and O. Babaoglu, “A Modular Paradigm for Building Self-Organizing Peer-to-Peer Applications,” in *Proceedings of the International Workshop on Engineering Self-Organising Applications*, 2004, pp. 265–282.
- [203] M. Jelasity, W. Kowalczyk, and M. van Steen, “An Approach to Massively Distributed Aggregate Computing on Peer-to-Peer Networks,” in *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2004, pp. 200–207.
- [204] M. Jelasity and A. Montresor, “Epidemic-Style Proactive Aggregation in Large Overlay Networks,” in *Proceedings of the 24th International Conference on Distributed Computing Systems*, 2004, pp. 102–109.

- [205] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic Algorithms for replicated Database maintenance," in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing - PODC '87*, 1987, pp. 1–12.
- [206] R. van Renesse, D. Dumitriu, V. Gough, and C. Thomas, "Efficient Reconciliation and Flow Control for Anti-Entropy Protocols," in *Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware - LADIS '08*, 2008, pp. 1–7.
- [207] Hazelcast, "Hazelcast the Leading In-Memory Data Grid - Hazelcast.com." [Online]. Available: <https://hazelcast.com/> (Accessed on: 2017-05-02).
- [208] Atomix, "Atomix - Fault-tolerant Distributed Coordination Framework for Java 8." [Online]. Available: <http://atomix.io/atomix/> (Accessed on: 2018-01-17).
- [209] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in SDN-based core networks," in *Proceedings of the 25th International Teletraffic Congress (ITC)*, 2013, pp. 1–9.
- [210] B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement Problem," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 7–12.
- [211] C. Whelan, G. Harrell, and J. Wang, "Understanding the K-Medians Problem," in *Proceedings of the International Conference on Scientific Computing (CSC), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp)*, 2015, pp. 219–222.
- [212] V. Arya, N. Garg, R. Khandekar, K. Munagala, V. Pandit, and V. Pandit, "Local Search Heuristic for k-median and Facility Location Problems," in *Proceedings of the thirty-third annual ACM symposium on Theory of computing - STOC '01*, 2001, pp. 21–29.
- [213] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms - David P. Williamson, David B. Shmoys - Google Books*, 1st ed. Cambridge University Press, 2011.

- [Online]. Available: <http://www.designofapproxalgs.com/book.pdf>
- [214] D. Hock, M. Hartmann, S. Gebert, T. Zinner, and P. Tran-Gia, "POCO-PLC: Enabling dynamic pareto-optimal resilient controller placement in SDN networks," in *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2014, pp. 115–116.
- [215] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4–17, 2015.
- [216] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic Controller Provisioning in Software Defined Networks," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, 2013, pp. 18–25.
- [217] S.-C. Lin, P. Wang, I. Akyildiz, and M. Luo, "Towards Optimal Network Planning for Software-Defined Networks," *IEEE Transactions on Mobile Computing (Early Access)*, pp. 1–1, 2018.
- [218] G. Ishigaki, R. Gour, A. Yousefpour, N. Shinomiya, and J. P. Jue, "Cluster Leader Election Problem for Distributed Controller Placement in SDN," in *Proceedings of the IEEE Global Communications Conference - GLOBECOM 2017*, 2017, pp. 1–6.
- [219] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier, "Inter-controller traffic in ONOS clusters for SDN networks," in *Proceedings of the IEEE International Conference on Communications (ICC)*, 2016, pp. 1–6.
- [220] H. R. Faragardi, H. Fotouhi, T. Nolte, and R. Rahmani, "A Cost Efficient Design of a Multi-sink Multi-controller WSN in a Smart Factory," in *Proceedings of the IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2017, pp. 594–602.

REFERENCES

- [221] “k-means Clustering - MATLAB kmeans.” [Online]. Available: <https://www.mathworks.com/help/stats/kmeans.html> (Accessed on: 2018-04-24).
- [222] Y. Zhang, E. Ramadan, H. Mekky, and Z.-L. Zhang, “When Raft Meets SDN,” in *Proceedings of the First Asia-Pacific Workshop on Networking - APNet'17*, 2017, pp. 1–7.
- [223] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, “ZooKeeper: Wait-free Coordination for Internet-scale Systems,” in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, 2010, pp. 1–11.
- [224] ONOS, “ONOS.” [Online]. Available: <https://wiki.onosproject.org/> (Accessed on: 2016-08-22).
- [225] R. C. A. Alves, D. A. G. Oliveira, G. A. Núñez, and C. B. Margi, “IT-SDN: Improved architecture for SDWSN,” in *Proceedings of the XXXV Brazilian Symposium on Computer Networks and Distributed Systems*, 2017. [Online]. Available: <http://www.larc.usp.br/~cbmargi/it-sdn>