Performance Evaluation of a Software-Defined Wireless Sensor Network

Tubagus Rizky Dharmawan

Master of Science Thesis

Delft University of Technology

MSc Embedded Systems

Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science
MSc Embedded Systems

Performance Evaluation of a Software-Defined Wireless Sensor Network

Tubagus Rizky Dharmawan
4501225

November 27, 2017

Acknowledgement

All praises and thanks to Allah SWT, the Gracious, and the Merciful

Time flies but memories last forever. In TU Delft, I had many memorable experiences, both happiness and difficulties were faced by me during my study in here. I learned not only how to solve a problem, but also how to balance academic and social life. I realize that being a master student of TU Delft is complicated. There are many challenges and obstacles, but I am very grateful for all the opportunities and challenges. Through this acknowledgment, I would like to thank all who have helped and supported me.

First of all, all praises and thanks to Allah SWT, who gives me strength and patience so that I can accomplish my study in TU Delft. I would like to express my gratitude to my supervisor Dr.ir. Fernando Kuipers who gave me direction, guidance, and evaluation. Dr. Vijay Rao who gave me a valuable insight and then my parents and my big family who always support and pray for me.

Special thanks to ES-Indonesia: Andri, Bontor, Arif, Angga, Priadi, and Ridho for their support to strengthen each other and fight together. Abi and Andre who give a valuable feedback for improving this thesis. Keluarga Muslim Delft (KMD) as a place to give me a spiritual power and TU Delft-Indonesia 2015 for all our unforgettable memories.

Last but not least, I would like to gratefully acknowledge LPDP (Indonesia Endowment Fund for Education) for giving me a scholarship and an opportunity to pursue my Master's Degree in TU Delft.

Delft, November 2017
Tubagus Rizky Dharmawan

Abstract

Software-Defined Networking (SDN) *potentially* can improve the flexibility and management of Wireless Sensor Networks (WSNs). To investigate the impact of SDN on WSN, in this thesis, we consider three Software-Defined Wireless Sensor Network (SD-WSN) frameworks, namely SDN-WISE, SDWN-ONOS, and TinySDN. After comparing these frameworks, the performance of TinySDN is evaluated in three different scenarios: homogeneous, heterogeneous, and dynamic networks. The Collection Tree Protocol (CTP) is used for the evaluations, and is subsequently compared with Rime data collection protocol in Contiki. Our performance evaluation is based on four metrics: Packet Delivery Ratio (PDR), packet duplication, duty cycle, and delay.

Our results show that the PDR for WSN and SD-WSN is relatively similar, that is, around 0.98 to 1 for homogeneous and heterogeneous networks, whereas in a dynamic network, the PDR of WSN decreases from 0.98 to 0.9. Compared to the WSN, the SD-WSN reduces both the average delay of *SDN sensor node* and the time the *SDN sensor node* is active to send packets in homogeneous and heterogeneous networks. However, implementing a centralized controller in a dynamic network may cause the SD-WSN performance decrease, which is indicated by the increase ratio of average delay from 2.03 to 2.3, whereas the increase ratio of average delay for the WSN is only around 1.6 to 1.98. The packet duplication level increases by 33% in the dynamic network when the number of *SDN sensor nodes* increases from 10 to 15.

The performance of SD-WSN in heterogeneous, homogeneous, and dynamic networks is relatively worse than WSN in terms of packet duplication and Rx duty cycle. SD-WSN, in addition, is not optimally implemented for dynamic conditions as the activity changes from the *SDN sensor nodes* will significantly affect the performance of SD-WSN. To reduce the load on the centralized controller, the clustering controllers are deployed to distribute the load on multiple controllers. The result shows that packet duplication and average delay in a dynamic network can be reduced by 13% and 57%, respectively. Clustering controllers provide more stability in terms of Rx duty cycle compared to before using clustering controllers.

Keywords *: SDN, WSN, Performance Evaluation, Collection Protocol*

Table of Contents

List of Figures

List of Tables

Glossary

API Application Programming Interface

DGRM Directed Graph Radio Medium

ETX Expected Transmission

HVAC Heating, ventilation, and air conditioning

IoT Internet of Things

LQI Link Quality Indicator

nesC Network Embedded Systems C

ONF Open Network Foundation

PDR Packet Delivery Ratio

QoS Quality of Services

RSSI Received Signal Strength Indicator

SDN Software Defined Networking

SDN ECCKN Software Defined Networking Energy Consumed uniformly-Connected K-Neighborhood

SDN-TAP SDN-TAP: An SDN-based traffic aware protocol for wireless sensor networks

SDN-WISE Software Defined Networking solution for Wireless Sensor Networks

SDWN Software Defined Wireless Networks

SDWN-ONOS Software-Defined Network Operating System for the IoT

SD-WSN Software-Defined Wireless Sensor Networks

SDWSN-IoT Software-Defined Wireless Sensor Networks and Internet of Things

UDGM Unit Disk Graph Medium

WARM WSN Application Development and Resource Management

WPAN Wireless Personal Area Network

WSN Wireless Sensor Network

Chapter 1

Introduction

Nowadays, Wireless Sensor Networks (WSN) play a significant role in our life. According to the article from Business Week Online, WSN is considered as one of the most important technologies for the 21st century [35]. Currently, many technologies use WSN as their main feature, such as smart buildings and smart homes. These technologies utilize sensor networks to monitor the energy consumption of all appliances with the aim of achieving efficient energy utilization [19]. WSNs are also widely used in industrial areas. ON World reported that the use of wireless devices in industries have increased by 553% between 2011 and 2016. Up to 24 million wireless devices are deployed worldwide, and around 39% among these are used for the applications that are only possible with WSN [7].

There is a forecast that by 2020, fifty billion of things will be deployed worldwide and connected to the Internet [29]. With the fast technological development of sensor networks, WSN becomes the main component for the Internet of things (IoT) [7]. It enables physical objects to perform a task by having them communicate with each other, to share data and to coordinate decisions [13]. For instance, by integrating the sensor networks in Heating, Ventilation, and Air conditioning (HVAC) in building systems to create a smart technology which complies with the user requirements.

The deployment of WSN devices in the future will increase exponentially due to the needs for extensive observation in industries, automotive, transportation, and other infrastructures [7]. As a result, the large-scale of WSN will generate data from today's exabyte (1018 bytes) level to zettabyte (1021 bytes) level [7]. An estimate by IDC statistics and forecast mentions that, in 2020, the global data volume will reach 35 zettabyte [7]. The increasing demands for WSN and the diverse needs for WSN services make the management system of WSN more difficult. Therefore, there is a crucial need to build an open management system and flexible framework to tackle this bottleneck.

Software Defined Networking (SDN) is one of the promising technologies that has the potential to solve the challenges in the management system of WSNs. Initially, the idea of SDN is proposed to handle several problems in the wired network, such as the complexity of traditional internet protocol network and the difficulty of configuring the network behavior

[21]. SDN emerges to tackle these issues by decoupling the control logic of the network (control plane) from the network switches (data plane). This concept creates a centralized control logic where the policy of the network is defined by the control plane, as shown in Figure 1-1.

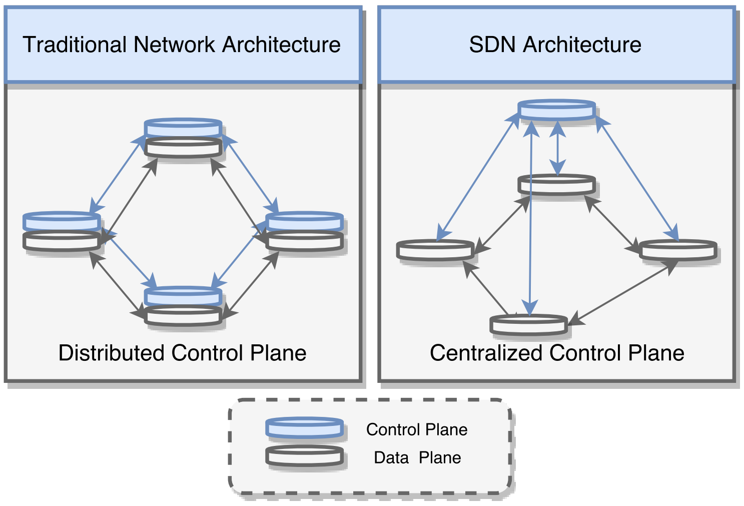

Traditional Network Architecture
SDN Architecture
Distributed Control Plane
Centralized Control Plane
Control Plane
Data Plane

Figure 1-1: Traditional Network VS Software Defined Networking

Several companies like Google, Facebook, Microsoft, Verizon, have collaborated and built the Open Network Foundation (ONF) consortium [21]. ONF is created to promote SDN as a solution for creating a more flexible network infrastructure. They consider SDN as a solution to provide more dynamic, cost-effective, and adaptable large network infrastructure [41]. Google have successfully utilized SDN, improving the efficiency of its data center connection around the world by 2-3 times compared to the traditional network infrastructure [20].

The implementation of SDN on the wired network delivers various advantages, such as centralized network provisioning and simpler hardware management [40]. These become the reasons why SDN is considered as a promising technology to solve the problems in WSN management system, such as resource-constrained system, application-specific architecture, and Quality of Service (QoS) maintenance [28]. By decoupling the control logic from the sensor node, the centralized controller is expected to reduce the complexity of the sensor network management [28, 18, 4]. SDN abstracts the entire sensor network so that the implementation of new network policy and the handling of the user requirements related to the QoS of the WSN will be easier [28].

1-1 Motivation and Problem Description

1-1-1 Motivation

Initially, SDN was developed for solving a problem in the traditional wired network in which the data plane, which is responsible for forwarding packets, and the control plane, which is responsible for routing, are coupled to every network device (switch or router). Having a control plane and a data plane in every network device means all network devices define their own decision in a distributed fashion [22].

Using a distributed control system in a large traditional network creates challenges for the network administrator since the configuration of the network devices should be configured manually [22]. This adversity led to the idea of separating the control plane from the data plane, that is, SDN.

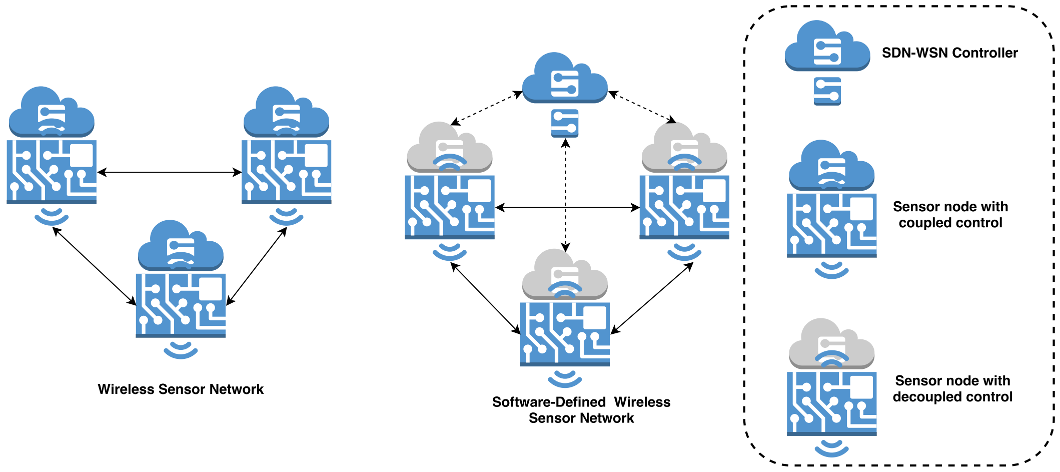

Wireless Sensor Network
Software-Defined Wireless Sensor Network
SDN-WSN Controller
Sensor node with coupled control
Sensor node with decoupled control

Figure 1-2: The implementation of SDN concept in WSN

SDN creates the network device (switch or router) become a "dumb" device, which is responsible only for forwarding the packet. The control function itself is performed by a single centralized entity, that is, SDN controller. SDN controller is responsible for defining the behavior of all of the network devices and managing the entire network. The centralized paradigm in SDN allows routing policies to be easily deployed and provides innovation opportunity in the network [22].

The concept of a distributed control system in the traditional wired network has similar characteristics with WSN, as shown in Figure 1-2. Each sensor node has their own "brain", and thus the determination of forwarding data to other nodes is independent. One of the objectives of a distributed control system in WSN is to avoid a single point of failure since WSN is usually used in extreme and remote areas, such as in forests and mountains. However, the characteristic of a distributed control system in WSN makes the network development in large WSN difficult to be handled, since it requires (re)programming individual sensor node to adapt to a dynamic environment. The application programmers need a higher-level abstraction in order to ease their task, but the current design of WSN is not designed to provide higher-level abstraction [38]. Therefore, a dynamic policy change on WSN will be difficult [25].

Due to the flexibility possessed by SDN in which the network behaviour is defined by a centralized controller, researchers have become interested in developing SDN technology in WSN. They are interested with the benefits of SDN which simplifies network management by decoupling the control plane from the data plane in the sensor nodes [21]. Therefore, the sensor node, which acts as the data plane, simply forwards the packet, while the network intelligence, routing, and QoS control are defined by the centralized controller, as shown in Figure 1-2.

Although SDN is predicted to solve problems in the WSN management system, research on its effects on WSN is still in its infancy. Combining these two technologies does not guarantee that the performance of WSN will be improved. Therefore, further research is needed to investigate the impact of SDN implementation in WSN to the network performance.

1-1-2 Problem Description

The integration between SDN and WSN, which can be denoted as Software Defined Wireless Sensor Network (SD-WSN), is expected to improve the flexibility of WSN management, simplify the WSN architecture, and support innovations via network programmability [28, 18, 4, 9]. However, based on a literature survey and preliminary experiment for the SD-WSN framework, there are two major problems in the implementation of SDN in WSN: lack of research and immature platform.

Lack of Research

SDN cannot be guaranteed as one-fits-all solution for WSNs management as research on SD-WSN is still in its infancy. Moreover, most of the research only discusses the concept of integrating SDN into WSN without extensive evaluation or performance implication [39, 15, 25, 49, 6, 1].

The research of SD-WSNs seems to be disconnected from WSN research [33]. This condition leads to an inadequate understanding of challenges faced when implementing SDN on WSN. Most of the challenges in SD-WSN are due to the limited resources that WSNs possess, such as low data rates, limited performances, limited energy resources, and unreliable links. The researchers tried to tackle these challenges by modifying the architecture of SDN. However, the solution is still a proof of concept and is not specified to solve real problems in WSN.

To the best of author's knowledge, there is no research which performs an extensive evaluation of SD-WSN. There is limited information on the previous work whether SDN gives a better performance or not. In addition, the current research of SD-WSN cannot explain the impact of SDN on WSN performance (network lifetime, packets lost, quality of service, and adaptation to dynamic environment). This should be the main concern for the researcher before proceeding SD-WSN to the next stage.

Immature Platform

To the best of author's knowledge, there are three open-source frameworks that can be used for implementing SDN on WSN, namely SDN-WISE [14], SDWN-ONOS [1], and TinySDN [31]. These platforms share the same objective, that is, to demonstrate that the concept of SDN can be applied to WSN. However, these frameworks have several drawbacks, such as limited documentation and lack of community support to develop these frameworks. In SDN-WISE, the sensor node cannot be implemented in real sensor network devices, e.g., TelosB, MICAz, Zolertia Z1, since the sensor node is custom-built, and thus the code from these frameworks does not comply with the existing WSN firmware. Consequently, SDN-WISE cannot be integrated with the existing WSN.

SDWN-ONOS made a breakthrough by using an existing SDN controller on the wired network, namely Open Network Operating System (ONOS). In order to make ONOS comply with SD-WSN network, SDWN-ONOS modified the architecture of ONOS and used the custom WSN devices from SDN-WISE, namely SDN-WISE Emulated Mote and SDN-WISE Emulated sink. However, these devices are not designed based on IEEE 802.15.4 standard. Thus, this framework cannot be implemented in the real WSN device as well.

SDN-WISE, SDWN-ONOS, and TinySDN are considered unable to provide a comprehensive solution to handle the problems in WSN. They have not been able to evaluate the significance of implementing SDN in WSN. Therefore, further testing in more scenarios, benchmarking with existing WSN, and more development are required.

1-2 Research Objective

The main objective of this thesis is **"to evaluate and identify the performance challenges for wireless sensor networks when integrated with Software Defined Networking"**. With this regard, the following objectives are defined:

- To figure out whether an SD-WSN could perform as good as a decentralized WSN
- To analyze the results of SD-WSN simulation against WSN performance metrics (duty cycle, delay, packet delivery ratio, and packet duplication) in several scenarios (homogeneous network, heterogeneous network, and dynamic network)
- To present a solution to make SDN suitable for functioning in WSN management

1-3 Research Questions

Based on the research objective described, the research questions for this thesis can be elaborated as the following:

- What are the trade-offs between duty cycle, delay, packet delivery ratio and packet duplication when WSN uses SDN for management?
- What kind of improvements could SDN bring for WSN compared to an existing solution?

1-4 Methodology

In this thesis, the performance of software defined wireless sensor network is evaluated using one of the SDN frameworks for WSN, that is, TinySDN [31]. TinySDN is an open-source framework for SD-WSN and is designed to be hardware independent. The TinySDN framework is applied in both *SDN controller node* (control plane) and *SDN sensor node* (data plane) through TinySDN API. The Collection Tree Protocol (CTP) is used for collecting data in SD-WSN [17]. Afterwards, the performance of SD-WSN, which uses CTP, will be compared with WSN, which uses Rime data collection protocol in Contiki [10]. Figure 1-3 shows the steps that shall be followed.

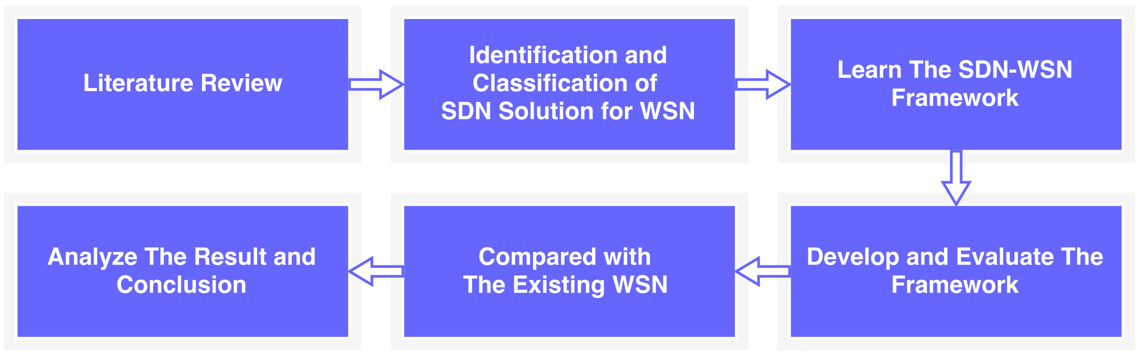

Literature Review
Identification and Classification of SDN Solution for WSN
Learn The SDN-WSN Framework
Develop and Evaluate The Framework
Compared with The Existing WSN
Analyze The Result and Conclusion

Figure 1-3: The Methodology

- Literature Review: In the initial stage, the survey paper which is related to the topic of SDN, WSN, and IoT is read. From the survey paper, the information, the main issues, and the challenges of SD-WSN are summarized to figure out a state-of-the-art of SD-WSN.
- Identification and classification of the SDN solution for WSN: At this stage, all the relevant papers related to SD-WSN are divided into several categories: advantages and disadvantages, features, simulations, availability, and performance evaluation.
- Learn the SD-WSN framework: After identifying some papers on SDN solution for WSN, there are three open-source frameworks that can be used, namely SDN-WISE, SDWN-ONOS, and TinySDN. Afterwards, these framework are tested and analyzed to figure out the extent to which these frameworks can be used to solve the problems in SD-WSN.
- Develop and evaluate the framework: After preliminary experiments using these three frameworks, TinySDN is selected for further analysis.
- Compared with the existing WSN: Collection tree protocol (CTP) is used as the protocol that is implemented on TinySDN. This framework is evaluated based on four performance metrics and three scenarios. Duty cycle, delay, packet delivery ratio, and packet duplication are used as an indicator for SD-WSN performance metrics. To vary the network environments, homogeneous, heterogeneous, and dynamic network scenarios are evaluated. The results from SD-WSN experiments are compared with the experimental results from the Rime data collection protocol in Contiki.
- Analyze the result and conclusion: In the final stage, the result between SD-WSN and an existing WSN are analyzed. This result is used to answer the objectives and the conclusions about performance challenges for wireless sensor network when integrated with software defined networking.

1-5 Thesis Structure

This thesis is organized into five chapters. Chapter 1 provides an introduction to WSN and SDN; this chapter explains an overview of the problems faced by the WSN and the solutions offered by SDN. This chapter also discusses the methodology to answer the research questions. Chapter 2 describes the summary of the literature review and the comparison of the SDN solutions for WSN. Chapter 3 covers the implementation of centralized controller using SDN framework into WSN and then several test scenarios will be introduced. Chapter 4 discusses the performance evaluation of SDN for WSN based on the several test scenarios, and the result is then compared with an existing WSN solution. Finally, Chapter 5 concludes the overall thesis and provides possible future work.

Chapter 2

SDN Solutions for WSN

The idea of SDN is based on the separation of the control plane from the data plane. This concept creates a centralized controller which can manage and control the overall behavior of the network. The network devices such as sensor nodes simply forward or drop a packet based on the instruction from the controller. SDN allows network configuration to be performed globally as opposed to a distributed approach which requires individual configuration [28]. Figure 2-1 shows the principles of dividing the network where the control plane determines the traffic route and the data plane forwards the traffic packet.

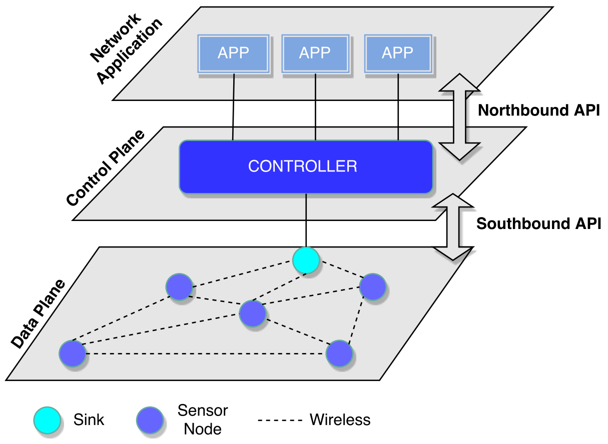


Figure 2-1: SD-WSN Architecture

SDN provides convenience in terms of the regulation and control of the network. This simplicity allows SDN to be implemented in many technologies. Some journals and researchers propose SDN as a solution to tackle several problems in WSN, such as energy efficiency [45, 48, 47] and network management [12, 31]. However, the results are still limited and most of them only propose a concept of SD-WSN without an extensive evaluation [39, 15, 25, 49, 6, 1]. The differences regarding characteristics and requirements between wired networks and wireless sensor networks make implementing SDN into WSN challenging since WSN has many limitations, such as limited power, low communication capabilities, and small processing capabilities [15].

Although SD-WSN research is still in its infancy, several researchers have initiated their research to propose an ideal solution for SD-WSN. The research of SDN for WSN is started by two early adopters: Sensor OpenFlow [25] and SDWN [9].

2-1 The Early Adopters of SD-WSN

Luo et al. [25] proposed Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks. This research is one of the early adopters of SDN on WSN. The focus of Sensor OpenFlow is to propose a concept of Southbound API as this component is very crucial for sending an instruction or receiving a request between the data plane and the control plane. The idea of Sensor OpenFlow is inspired by one of the de-facto protocols in SDN, namely OpenFlow [26]. However, OpenFlow is designed for a wired protocol, and thus this protocol is not comply with WSN. To address this issue, Sensor OpenFlow proposes a modification based on the OpenFlow version v1.3.0 so that the Sensor OpenFlow can be used as a wireless sensor communication protocol between the data plane (sensor nodes) and control plane in SD-WSN.

Sensor OpenFlow's concept claims to be able to support multiple applications, flexible, and improve the management system in WSN. However, several things are missing in this paper, such as the explanation of the control plane, specifically whether the control plane belongs to a sensor network or non-sensor network. Sensor OpenFlow also does not discuss the specification and the performance evaluation of its system. Therefore, Sensor OpenFlow is difficult to be implemented into real devices.

Costanzo et al. [9] proposed a concept of SD-WSN, namely SDWN (Software Defined Wireless Networks). SDWN is also one of the early adopters of SD-WSN, but the approach is different with Sensor Open Flow. Unlike Sensor OpenFlow, SDWN proposes an API for the communication interface between sensor node and controller, namely flow table. SDWN tries to analyze the benefits of SDN in wireless infrastructure networking and to validate the statement that SDN brings simplification and flexibility of the network management.

The main contribution of SDWN is to propose a concept of SDWN architecture while considering the characteristics of WSN which are based on IEEE 802.15.4, such as supporting duty cycles and in-network data aggregation to reduce the energy consumption. The proposed architectures of SDWN consist of two components: generic node and sink node. The generic node acts as the data plane where the forwarding layer is executed. It is also responsible for handling packets as specified by the controller. The sink node acts as the control plane in which the rules for the data plane is defined.

SDWN proposes an architecture for SD-WSN. However, this research has not proven to solve any problems in the management of WSN. Moreover, SDWN could not answer the main reason why SDN should be implemented in WSN, since there is no performance evaluation to validate the benefits of SDWN. SDWN can be considered as the first step towards SDN solutions for WSN, since many papers refer to SDWN as the foundation for their researches, such as SDN-WISE [14], SDWN-ONOS [1] and SDN-TAP [12].

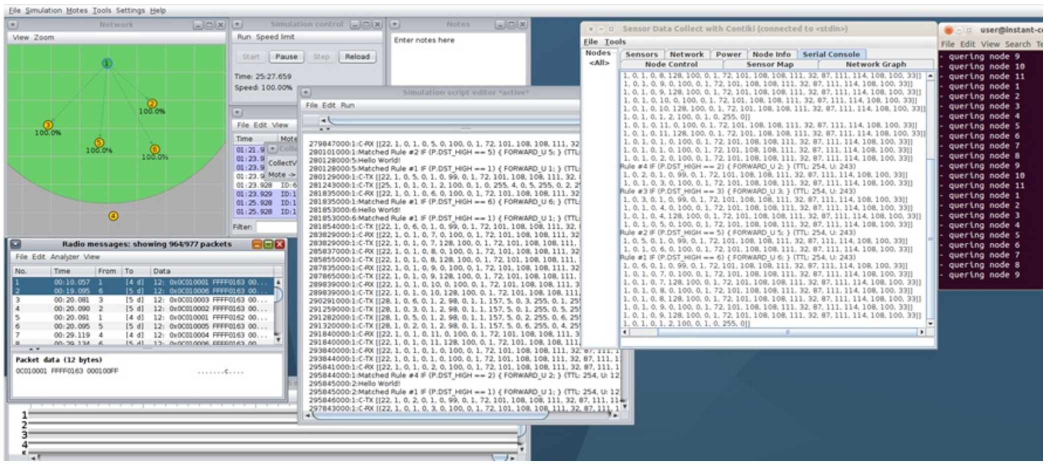
2-2 The Framework of SD-WSN

To tackle a few shortcomings of Sensor OpenFlow and SDWN related to the (lack of) performance evaluation, an API for SD-WSN, and the architecture of SD-WSN, Galluccio et al. [14] proposed SDN-WISE (Software Defined Networking solution for Wireless Sensor Networks), Anadiotis et al. [1] proposed SDWN-ONOS (Software-Defined Open Network Operating System), and Oliveira et al. [31] proposed TinySDN (TinyOS-based SDN), respectively.

2-2-1 SDN-WISE

The concept of SDN-WISE is based on the SDWN [9], but the architecture is modified in order to comply with IEEE 802.15.4. The architecture of SDN-WISE is divided into three main parts, namely sensor node, sink, and control plane. Figure 2-2 shows the preliminary experiment of SDN-WISE. The sensor node acts as the data plane that executes commands from the control plan to its device. The sink acts as a gateway between sensor nodes, and the control plane acts as a central network management.

WISE flow table is used as the communication protocol between the sensor nodes and the controller in SDN-WISE. WISE flow table contains a set of rules installed on each sensor node. To contact the controller, the sensor node needs a WISE Flow Table entry which indicates its best next hop towards the sink.



```
01:21.9
01:23.9
01:23.9
01:23.9
01:23.928  ID:6
01:23.929  ID:1
01:25.928  ID:1
01:25.928  ID:1
```

```
Radio messages: showing 964/977 packets
File  Edit  Analyzer  View
No.   Time        From   To      Data
1     00:10.057   1      [4 d]   12:  0x0C010001  FFFF0163  00...
2     00:19.095   6      [5 d]   12:  0x0C010006  FFFF0163  00...
3     00:20.081   3      [5 d]   12:  0x0C010003  FFFF0163  00...
4     00:20.090   2      [5 d]   12:  0x0C010002  FFFF0163  00...
5     00:20.091   1      [4 d]   12:  0x0C010001  FFFF0162  00...
6     00:20.095   5      [5 d]   12:  0x0C010005  FFFF0163  00...
7     00:29.119   4      [4 d]   12:  0x0C010004  FFFF0163  00...

Packet data (12 bytes)
0C010001  FFFF0163  000100FF                    .......c....
```

```
279847000:1:C-RX [[22, 1, 0, 1, 0, 5, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32
280101000:1:Matched Rule #2 IF (P.DST_HIGH == 5) { FORWARD_U 5; } (TTL
280128000:5:Hello World!
280128000:5:Matched Rule #1 IF (P.DST_HIGH == 1) { FORWARD_U 1; } (TTL
280129000:1:C-TX [[22, 1, 0, 5, 0, 1, 0, 99, 0, 1, 72, 101, 108, 108, 111, 32,
281243000:1:C-TX [[25, 1, 0, 1, 0, 1, 2, 100, 0, 1, 0, 255, 4, 0, 5, 255, 0, 2, 2
281835000:1:C-RX [[22, 1, 0, 1, 0, 6, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32
281835000:1:Matched Rule #1 IF (P.DST_HIGH == 6) { FORWARD_U 6; } (TTL
281853000:6:Hello World!
281853000:6:Matched Rule #1 IF (P.DST_HIGH == 1) { FORWARD_U 1; } (TTL
281854000:1:C-TX [[22, 1, 0, 6, 0, 1, 0, 99, 0, 1, 72, 101, 108, 108, 111, 32,
283829000:1:C-RX [[22, 1, 0, 1, 0, 7, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32
283829000:1:C-TX [[22, 1, 0, 1, 0, 7, 128, 100, 0, 1, 72, 101, 108, 108, 111,
285837000:1:C-RX [[22, 1, 0, 1, 0, 8, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32
285855000:1:C-TX [[22, 1, 0, 1, 0, 8, 128, 100, 0, 1, 72, 101, 108, 108, 111,
287835000:1:C-RX [[22, 1, 0, 1, 0, 9, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32
287865000:1:C-TX [[22, 1, 0, 1, 0, 9, 128, 100, 0, 1, 72, 101, 108, 108, 111,
289839000:1:C-RX [[22, 1, 0, 1, 0, 10, 0, 100, 0, 1, 72, 101, 108, 108, 111, 3
289839000:1:C-TX [[22, 1, 0, 1, 0, 10, 128, 100, 0, 1, 72, 101, 108, 108, 111,
290291000:1:C-TX [[28, 1, 0, 6, 0, 1, 2, 98, 0, 1, 1, 157, 5, 0, 3, 255, 0, 1, 25
291259000:1:C-TX [[28, 1, 0, 3, 0, 1, 2, 98, 0, 1, 1, 157, 5, 0, 1, 255, 0, 5, 25
291282000:1:C-TX [[28, 1, 0, 5, 0, 1, 2, 98, 0, 1, 1, 157, 5, 0, 2, 255, 0, 6, 25
291320000:1:C-TX [[28, 1, 0, 2, 0, 1, 2, 98, 0, 1, 1, 157, 5, 0, 6, 255, 0, 4, 25
291840000:1:C-RX [[22, 1, 0, 1, 0, 11, 0, 100, 0, 1, 72, 101, 108, 108, 111, 3
291840000:1:C-TX [[22, 1, 0, 1, 0, 11, 128, 100, 0, 1, 72, 101, 108, 108, 111,
293840000:1:C-RX [[22, 1, 0, 1, 0, 1, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 1
293844000:1:C-TX [[22, 1, 0, 1, 0, 1, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 1
295841000:1:C-RX [[22, 1, 0, 1, 0, 2, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 1
295844000:1:Matched Rule #4 IF (P.DST_HIGH == 2) { FORWARD_U 2; } (TTL: 254, U: 12
295845000:2:Hello World!
295845000:2:Matched Rule #1 IF (P.DST_HIGH == 1) { FORWARD_U 1; } (TTL: 254, U: 12
295846000:1:C-TX [[22, 1, 0, 2, 0, 1, 0, 99, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 11
297843000:1:C-RX [[22, 1, 0, 1, 0, 3, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 1
```

```
 1, 0, 1, 0, 8, 128, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 9, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 9, 128, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 10, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 10, 128, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 1, 2, 100, 0, 1, 0, 255, 0]]
 1, 0, 1, 0, 11, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 11, 128, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 1, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 1, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 2, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
Rule #4 IF (P.DST_HIGH == 2) { FORWARD_U 2; } (TTL: 254, U: 243)
 1, 0, 2, 0, 1, 0, 99, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 3, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
Rule #3 IF (P.DST_HIGH == 3) { FORWARD_U 3; } (TTL: 254, U: 243)
 1, 0, 3, 0, 1, 0, 99, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 4, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 4, 128, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 5, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
Rule #2 IF (P.DST_HIGH == 5) { FORWARD_U 5; } (TTL: 254, U: 243)
 1, 0, 5, 0, 1, 0, 99, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 6, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
Rule #1 IF (P.DST_HIGH == 6) { FORWARD_U 6; } (TTL: 254, U: 243)
 1, 0, 6, 0, 1, 0, 99, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 7, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 7, 128, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 8, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 8, 128, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 9, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 9, 128, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
 1, 0, 1, 0, 1, 2, 100, 0, 1, 0, 255, 0]]
```

```
- quering node 9
- quering node 10
- quering node 11
- quering node 1
- quering node 2
- quering node 3
- quering node 4
- quering node 5
- quering node 6
- quering node 7
- quering node 8
- quering node 9
- quering node 10
- quering node 11
- quering node 1
- quering node 2
- quering node 3
- quering node 4
- quering node 5
- quering node 6
- quering node 7
- quering node 8
- quering node 9
```

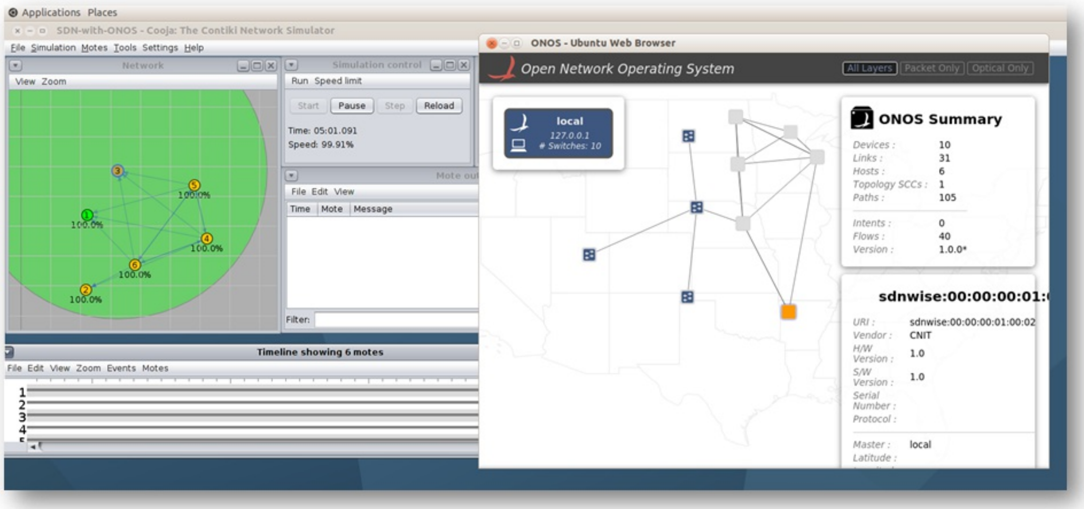
Figure 2-2: SDN-WISE

Compared to the previous work (SDWN and Sensor Openflow), SDN-WISE introduces the performance evaluation of their systems such as Round Trip Time (RTT), efficiency, and controller response time. However, the implementation of SDN-WISE has not been validated with WSN protocol. The idea of implementing SDN into WSN is to reduce the complexity of management system, but SDN-WISE is not able to prove whether integrating SDN on WSN will provide benefits or not. SDN-WISE also cannot be implemented in real sensor network devices, e.g. TelosB, MICAz, Zolertia Z1, since the sensor node in SDN-WISE is only supporting its system; thus, the code from this framework does not comply with the existing WSN devices.

2-2-2 SDWN-ONOS

SDWN-ONOS is a framework for SD-WSN which is developed from SDN-WISE. The aim of this framework is to create an SDN framework that can be implemented on IoT. In general, the components and the architecture in SDWN-ONOS are similar to SDN-WISE. However, in order to connect IoT and SDN, SDWN-ONOS adds a controller from wired network into SDN-WISE, namely ONOS [3]. Figure 2-3 shows the preliminary experiment of SDWN-ONOS.

ONOS has different characteristics with WSN since ONOS is used for the wired network. In order to apply ONOS into SDN-WISE, Anadiotis et al. modified the ONOS controller based on the SDN-WISE architecture. Although SDWN-ONOS can demonstrate that the SDN concept can be applied for IoT, the main problems faced by WSN are still not resolved by SDWN-ONOS. The implementation of SDWN-ONOS is not validated with the WSN protocol and is not able to prove whether integrating SDN on IoT will provide benefits or not. In addition, this framework cannot be implemented into real WSN devices as well, since the firmware of this sensor node is custom-built.


Applications Places
SDN-with-ONOS - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Network
View Zoom
100.0%
Simulation control
Run Speed limit
Start Pause Step Reload
Mote output
File Edit View
Time Mote Message
Filter:
Timeline showing 6 motes
File Edit View Zoom Events Motes
ONOS - Ubuntu Web Browser
Open Network Operating System
All Layers Packet Only Optical Only
local
127.0.0.1
# Switches: 10

Time: 05:01.091
Speed: 99.91%

ONOS Summary

Devices :	10
Links :	31
Hosts :	6
Topology SCCs :	1
Paths :	105
Intents :	0
Flows :	40
Version :	1.0.0*

sdnwise:00:00:00:01:

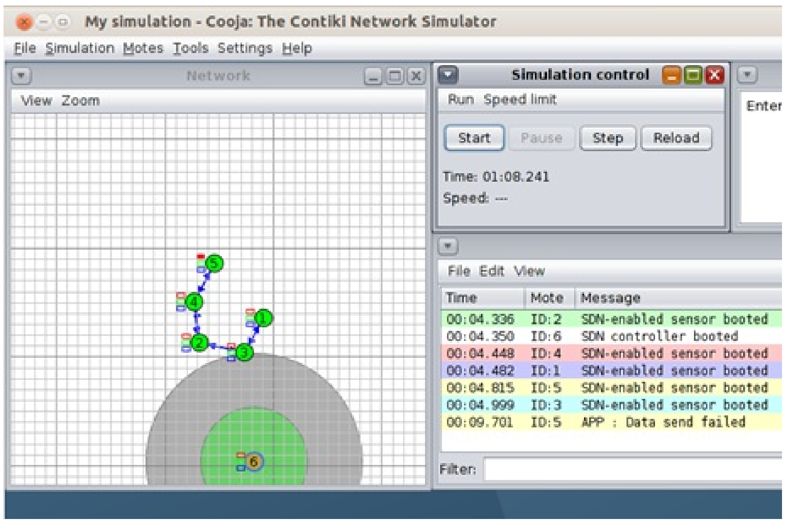
URI :	sdnwise:00:00:00:01:00:02
Vendor :	CNIT
H/W Version :	1.0
S/W Version :	1.0
Serial Number :	
Protocol :	
Master :	local
Latitude :	

Figure 2-3: SDWN-ONOS

2-2-3 TinySDN

Most of the existing frameworks of SD-WSN are not open source, and some of them are partially open such as SDN-WISE and SDWN-ONOS. De Oliveira et al. proposed TinySDN [31], an open-source framework to enable SDN controller for WSN.

TinySDN is based on TinyOS which consists of two main components: SDN-enabled sensor node, which acts as data plane, and SDN controller node, which acts as control plane where the intelligence of the network is programmed. As opposed to SDN-WISE and SDN-ONOS in which they cannot be implemented on real WSN hardware, TinySDN is designed to be hardware independent so that it can be implemented into many existing WSN devices, such as TelosB and Micaz. Figure 2-4 shows the implementation of TinySDN.



```
Time: 01:08.241
Speed: ---

Time        Mote   Message
00:04.336   ID:2   SDN-enabled sensor booted
00:04.350   ID:6   SDN controller booted
00:04.448   ID:4   SDN-enabled sensor booted
00:04.482   ID:1   SDN-enabled sensor booted
00:04.815   ID:5   SDN-enabled sensor booted
00:04.999   ID:3   SDN-enabled sensor booted
00:09.701   ID:5   APP : Data send failed
```

Figure 2-4: TinySDN

TinySDN provides several scenarios and evaluations to test its system and also inspires several papers which use this framework as a core architecture, such as Distributed SDWSN [30] and WARM [39]. However, the performance evaluation of TinySDN is limited, and this framework cannot answer the main doubt about how SDN can be beneficial for WSN since the evaluation performance is insufficient.

2-3 The Algorithm of SD-WSN

Apart from the framework of SD-WSN, some papers propose a centralized algorithm to solve several problems faced by WSN such as energy efficiency and supporting multiple applications in WSN. Zeng et al. [48] and Wang et al. [45] proposed algorithms which can be implemented in the centralized controller.

Zeng et al. proposed Multi-task SDSN (Energy Minimization in Multi-Task Software-Defined Sensor Networks) algorithm to support multiple applications by considering energy efficiency. To overcome the challenges in WSN energy, Multi-task SDSN utilizes a centralized algorithm that regulates the activation of the sensor and the work schedule of each sensor. Multi-task SDSN consists of a sensor control server (acting as the control plane) which defines the rules of the network by utilizing a centralized algorithm and sensor nodes (acting as data plane) which execute the rules from the control server.

Multi-task SDSNs provides a performance analysis of SDSNs using its proposed algorithm, such as effective sensing rate, rescheduling time and power efficiency. However, Multi-task SDSN is only focused on the design, implementation, and evaluation of their algorithm applied in SDSNs without considering the effect of this algorithm in WSNs. The description of the control plane and data plane behaviors are also insufficient, and thus the Multi-task SDSN will be difficult to implement in real WSN devices.

Wang et al. [45] proposed a similar approach as Zeng et al. They proposed an algorithm based on sleep scheduling mechanism, namely SDN-ECCKN (Software Defined Networking-

Energy Consumed uniformly-Connected K-Neighborhood). The idea of this algorithm is to reduce the total transmission time so the sensor node can sleep without reducing or affecting the performance of the network.

They claim that the SDN-ECCKN algorithm is better than EC-CKN (Energy Consumed uniformly- Connected K- Neighborhood) regarding network lifetime. SDN-ECCKN concept adopts the idea of SDN which removes the computation units from each node and then the computation is executed by the controller using EC-CKN to manage the entire network.

The architecture of SDN-ECCKN is divided into three main parts, namely controller, switch, and sensor node. The controller acts as the central computation which processes all network management to control the sensor node, the switch forwards the decision from the controller to sensor node or as a gateway between the sensor node and controller. The sensor node acts as the data plane which sense or collect information in the target area. The drawback of SDN-ECCKN is similar to Multi-Task SDSN where the description of the control plane and data plane behaviors are insufficient.

To summarize current research of SD-WSN, Table 2-1 compares several works which propose SDN solutions for WSN. The summary contains the features, simulator, availability, the evaluation and the performance comparison with existing WSN.

Table 2-1: SDN Solutions for WSN

Year	**SD-WSN**	**Features**	**Simulation**	**Availability**	**Perfomance Evaluation**	**Compared with existing WSN**
2012	Sensor OpenFlow [25]	Propose a concept of SD-WSN	No	No	No	No
2012	SDWN [9]	Propose a concept of SD-WSN	No	No	No	No
2014	Smart WSN-SDN [15]	Propose a concept of SD-WSN	No	No	No	No
2014	TinySDN [31]	Provide an SD-WSN framework	Cooja	Open	Response time, memory	No
2015	SDN ECCKN [45]	A centralized sleep scheduling algorithm	Matlab	No	Network Lifetime	Yes
2015	SDWN ONOS [1]	Provide an SDN-IoT framework	Cooja, Mininet	Partially Open	No	No
2015	Multi-Task SDSN [48]	A centralized algorithm to optimize energy efficiency	Gurobi Optimizer	No	Sensing rate, rescheduling time, power efficiency,	No
2015	SDN-WISE [14]	Provide an SD-WSN framework	Cooja	Partially Open	RTT, efficiency, response time	No
2016	Routing SDWSN [47]	Propose an energy efficient algorithm for SD-WSN	Matlab	No	Network Lifetime	Yes
2016	WARM [39]	Provide an SD-WSN framework	Cooja	No	Comm. overhead, memory	No
2016	SDN-TAP [12]	Provide an SD-WSN framework	Cooja	No	Delay, packet loss	No
2016	SDWSN-IoT [32]	Propose a concept of SDN-IoT	No	No	No	No

Chapter 3

Implementation

In this chapter, our performance evaluation setup of SDN framework for WSN is discussed, including TinySDN as the core component for the SD-WSN framework, data collection protocol, SD-WSN and WSN work flows, test scenario and the performance metrics for evaluating SD-WSN.

3-1 Framework Overview

In general, the framework of this thesis is divided into five main blocks, shown in Figure 3-1, namely nesC, TinyOS, TinySDN, TelosB, and Cooja. A brief description of each block is as follows:

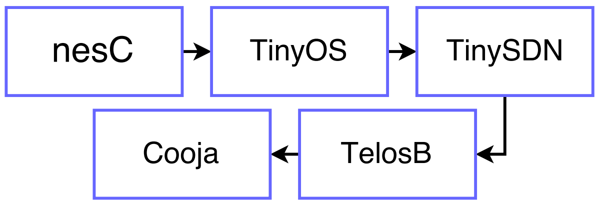


Figure 3-1: Framework Overview

- nesC: Network embedded systems C (nesC) [16] is a programming language used to create an application for TinyOS. This programming language is an extension of the C programming language with features to minimize the code size and random access memory (RAM). C and nesC have similarities, but the difference in nesC is that every program is formed by components, consisting of modules and configurations. The main task of the module is to declare the variables and the functions, while the main task of configuration is to connect components together (wiring).

  In short, to create an application on nesC, three files are required: Makefile, configuration file, and module file. The process that must be passed to produce an application from nesC is as follows: First, the nesC file is compiled using the nesC compiler, producing the app.c file. Afterwards, this C file is compiled with a native gcc compiler. As TelosB sensor node is used in this thesis, the msp430-gcc is used to compile and produce the binary file, that is, main.exe, which subsequently will be installed in the actual mote (TelosB or Micaz).

- TinyOS [23] version 2.1.2 is used as the operating system (OS) for our implementation. This OS is designed for wireless devices which have limited computational power and memory such as wireless sensor networks, personal area networks (PAN), and smart meter. TinyOS is not tied to certain devices, and thus can be implemented on various generic platforms, such as MicaZ, TelosB, TinyNode, and Zolertia Z1. Moreover, TinyOS supports various types of sensors that provide ease in the development of the new platform.

  This OS is written in the nesC programming language where the components are connected to each other by interfaces. This mechanism allows users to create components that can be made independent of hardware on various platform.

  To facilitate application usage and development, TinyOS provides abstraction, such as radio communication, timers, storage, and APIs which support various features, e.g. transmitting a packet, sensing sensor data, and communicating between each sensor node.

- TinySDN [31] version 0.2 is used as the framework for SD-WSN, it consists of two main components: *SDN sensor node* and *SDN controller node.* The *SDN sensor nodes* act like switches on SDN for a wired network which works only to forward packets. The *SDN controller node* acts as the control center which defines the behavior of the network.

- TelosB: TelosB [42] is a sensor node from Memsic which is used in our framework. This mote is compatible with the TinyOS and consists of MSP430 microcontroller, CC2420 radio chip, and several sensors, e.g. light sensor, temperature, humidity. In our framework, the TelosB is emulated in the Cooja simulator and used as SD-WSN devices for *SDN sensor node* and *SDN controller node.*

- Cooja [34]: In this thesis, Cooja is used as a network simulator to evaluate SDN framework on SDN. Cooja is an open-source network simulator. It can be used to emulate various types of sensor nodes. The code used in Cooja can be implemented on real sensor devices as the firmware used in Cooja is compatible with the firmware on the WSN devices (TelosB, Micaz, and Zolertia Z1)

  Cooja allows large and small networks of motes to be simulated. It is able to analyze the behavior of sensor networks. This simulator is useful to test the application code before running it on real hardware. Cooja is used to verify the behavior of our simulated network under several conditions and scenarios.

3-2 Architecture

In general, the architecture of SD-WSN is divided into two main components, shown in Figure 3-2, namely control plane and data plane. The architecture of this framework is implemented based on the TinySDN architecture [31], where the description for each component is as follows:

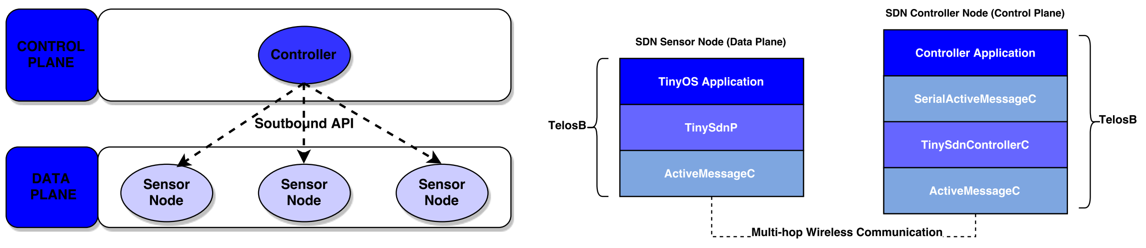


Figure 3-2: SD-WSN Architecture

3-2-1 Control Plane

In traditional WSN, the control plane and the data plane is coupled to each sensor node. In other words, each sensor node has their control plane, so the determination of forwarding data to other nodes is independent. In our architecture, the control plane and data plane is decoupled. Thus, the task of control plane and data plane are defined in different entities.

In the new control plane, the control plane works as one entity, namely *SDN controller node* which acts as the sink. The task of the sink is not only collecting the data from the sensor node but also deciding the routing path for each sensor node and managing the network flows. In contrast to the traditional WSN, the sink acts only for collecting information from each sensor node and forwarding all packets from each sensor node to the user.

In the control plane (*SDN Controller Node*), there are four sub-components: *Controller Application*, *SerialActiveMessageC*, *TinysdnControllerC*, and *Active MessageC*. The relation of each sub-component is depicted by Figure 3-3, and the function of each sub-component is as follows:

- *Controller Application*: This sub-component contains the control plane logic, such as managing network flows and deciding the routing path for each *SDN sensor node.*
- *SerialActiveMessageC*: The main task of this sub-component is to forward received packets from *SDN sensor node* to *SDN controller node* application or from *SDN controller node* application to the network.
- *TinysdnControllerC*: This sub-component is used for adapting the messages between *SDN controller node* and *SDN sensor node.*
- *ActiveMessageC*: This sub-component is a part of the TinyOS component that is responsible for managing and providing the radio module of the *SDN sensor node.* It is used for handling all tasks related to wireless communication.

3-2-2 Data Plane

In the traditional WSN, the functions of sensor nodes involve finding a route, sensing, and forwarding packets. In this architecture, the functions of the sensor node are reduced, so the

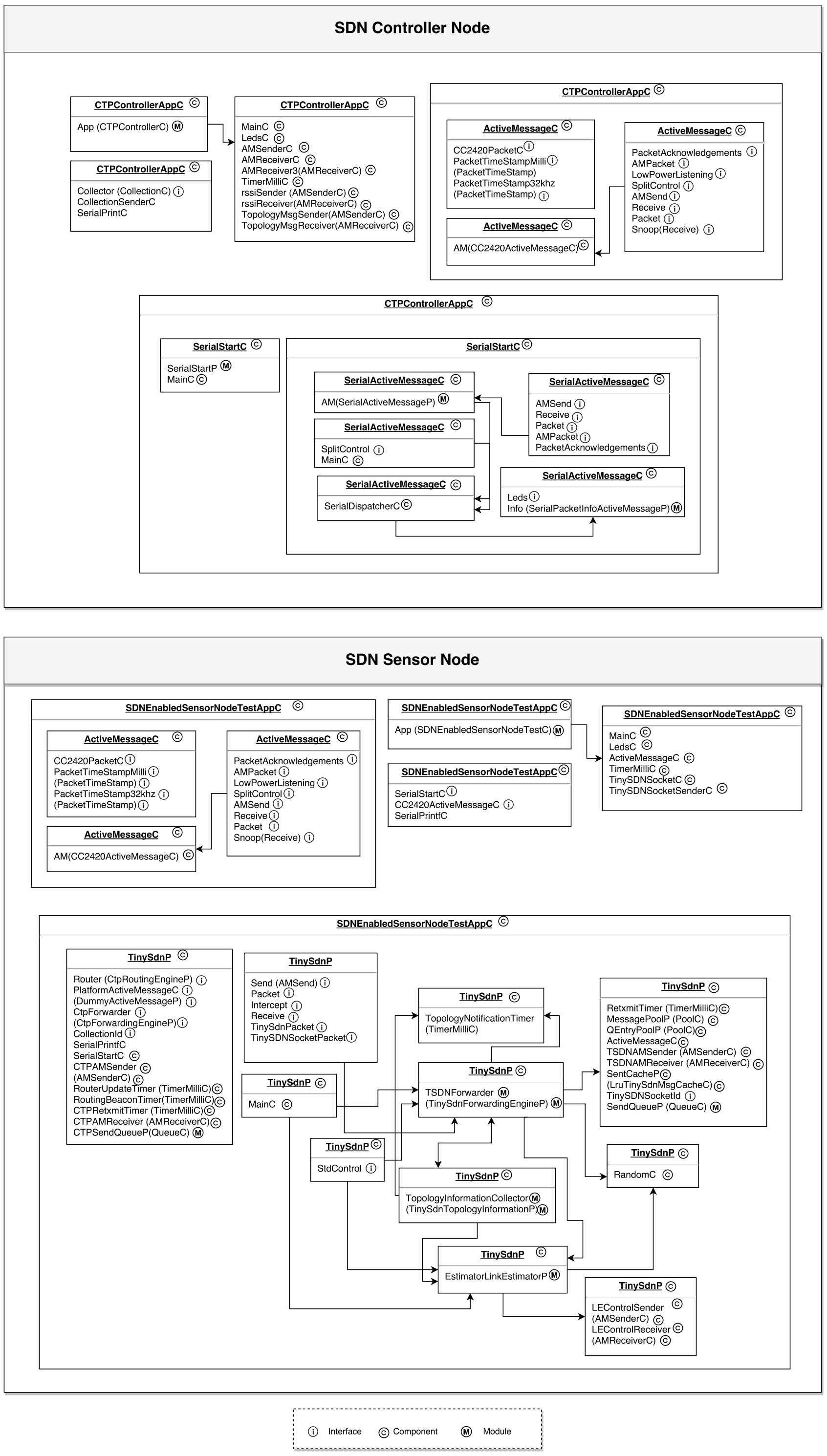


Figure 3-3: The Component Graph of *SDN Controller Node* and *SDN Sensor Node*

task is only forwarding information and sensing. The new entity of sensor node (only sensing and forwarding the packet) is defined by the data plane (*SDN sensor node*). In the data plane (*SDN sensor node*), there are three sub-components: *TinyOS Application*, *TinySdnP*, and *ActiveMessageC*. The relation of each sub-component is depicted by Figure 3-3 and the function of each sub-component is as follows:

- *TinyOS Application* is used for generating data packets and then to put the data packets on the network using application programming interface (API) provided by the TinySDN.
- *TinySdnP* is used for checking the received packet. If the incoming packet matches the flow table, then it performs the related action. Otherwise *TinySdnP* will send a flow setup request to the control plane (*SDN controller node*).
- The last sub-component in *SDN sensor node* is *ActiveMessageC*. This sub-component is responsible for communicating with the radio module/wireless communication of the sensor node. Furthermore, *ActiveMessageC* is also used for link quality estimation.

3-2-3 Southbound API

In the SD-WSN architecture, the southbound API is used to communicate between the *SDN controller node* and the *SDN sensor node*. The southbound API in SD-WSN is represented by a flow table which contains a set of instructions that will be executed by each *SDN sensor node*. Control flow is used to control traffic between *SDN controller node* and *SDN sensor node*, while data flow is used for application data traffic.

```
typedef struct {
  uint16_t dataFlowID;
  uint8_t  actionID;
  uint16_t actionParameter;
} flow_table_entry;
```

Listing 3.1: Flow Table Struct

Flow ID	Action	Action Parameter
0	Receive	-
1	Forward	5
2	Drop	-

Table 3-1: Flow Table

Table 3-1 denotes the flow table in SD-WSN, based on three data flow actions which are specified in TinySDN, namely *Forward*, *Receive*, and *Drop*. *Forward* means that the packet should be forwarded to the next hop of *SDN sensor node*; *Receive* denotes that the packet will be accepted by the *SDN sensor Node* or *SDN controller node*; and *Drop* defines that the packet is not used, thus the packet will be dropped. These actions will be categorized into three fields: Flow ID, to identify the flow which is used for the packets; Action, to specify the related action e.g. Forward, receive, and drop; Action Parameter, to specify the next hop. The structure of flow table is denoted by Listing 3.1.

3-3 Collection Tree Protocol (CTP)

The Collection Tree Protocol (CTP) [17] is the de-facto standard protocol in collection routing of WSN that makes the network look like a tree structure with the controller as root and the

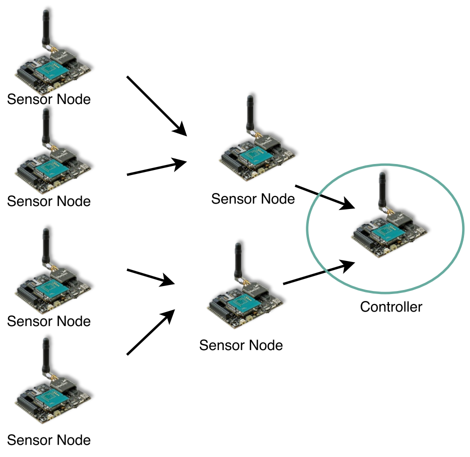


Figure 3-4: An example of data collection applications

sensor nodes at the perimeter of the network as the leafs. The concept of CTP is shown in the Figure 3-4. CTP is used for collecting the data from a network by assigning one node or several nodes as (a) tree root (s) in charge of collecting data generated by all the other sensor nodes in the network.

In order to create the route from the leafs to the root, the *SDN sensor node*s use routing gradient. Expected transmission (ETX) is a routing gradient used by CTP, where the lowest ETX will be selected as the path to the root (*SDN Controller Node*). This routing gradient can be defined as the number of transmissions an *SDN sensor node* has to make so that the packet can be acknowledged by the receiver. An *SDN sensor node*, for instance with an ETX of 2 can be assumed to be able to transmit a message to the other *SDN sensor nodes* after 2 transmissions. The following equations from 4 bits link estimator [11] show the process to obtain the ETX value.

Firstly, the packet reception ratio (PRR) is calculated based on the received beacons ($\mathrm{R_b}$) and failed beacons ($\mathrm{F_b}$).

$$PRR_{last} = \frac{R_b}{R_b + F_b} \tag{3-1}$$

Then, an exponentially weighted moving average (EWMA) function is used to calculate new PRR, with α as weighting factor between 0 and 1:

$$PRR_{new} = \alpha \times PRR_{old} + (1 - \alpha) \times PRR_{last} \tag{3-2}$$

and the PRR value is used to obtain the ETX value.

$$ETX = \frac{1}{PRR} \tag{3-3}$$

Finally, the ETX values are used to update the new ETX

$$ETX_{new} = \alpha \times ETX_{old} + (1 - \alpha) \times ETX_{last} \tag{3-4}$$

ETX values in CTP represent the 16-bit decimal fixed-point real number with a precision of tenths. An ETX value of 35, for instance, represents an ETX of 3.5, while an ETX value of 20 represents an ETX of 2.

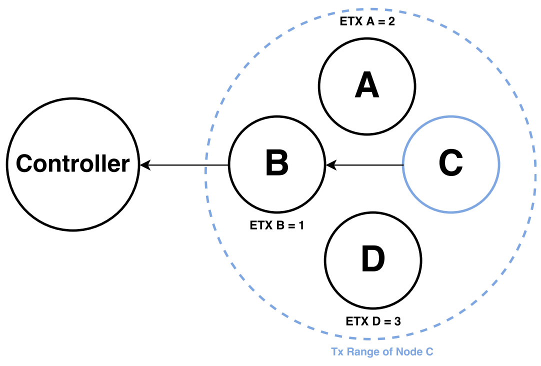


Figure 3-5: The Illustration of ETX Mechanism

Figure 3-5 illustrates the concept of ETX: there are four *SDN sensor nodes*: A, B, C, D and one *SDN controller node*. The *SDN sensor node* A, B, and D are within the transmission range of an *SDN sensor node* C. The *SDN controller node* has an ETX of 0, and *SDN sensor node* B who is the nearest node to the *SDN controller node* has an ETX of 1, which means when *SDN sensor node* C wants to send a packet to the *SDN controller node*, *SDN sensor node* C will use the node with the lowest ETX, that is *SDN sensor node* B, as the next hop. Overall, the aim of the ETX metric is to reduce the number of transmissions made by nodes. In order to keep the best routes up to date, each *SDN sensor node* broadcast periodic beacons to update the ETX value [24].

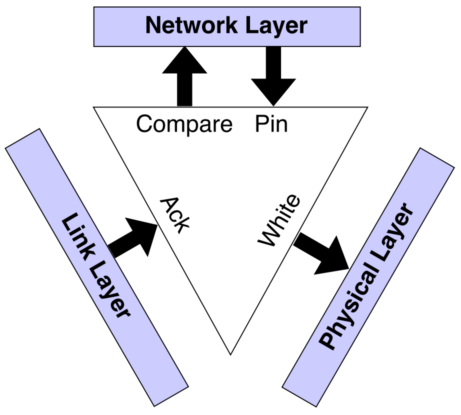


Figure 3-6: A Four-Bit Link Estimator [11]

The link estimator in CTP is using a four-bit (4B) link estimator [11] to assess the quality of its link. It uses information from three Open System Interconnection (OSI) layers, namely physical layer, data link layer, and network layers. Figure 3-6, shows the interfaces of 4B link estimator. These interfaces represent 4 bits of information: 2 bits from the network layer, namely pin bit and compare bit, 1 bit from the data link layer, namely ack bit, and 1 bit from the physical layer namely white bit.

The physical layer is used to denote whether the received packet experiences few errors or not. If the white bit is set, the channel quality is good. The link layer is used to indicate whether the acknowledgment packet is received or not. If the ack bit is set, the packet has been acknowledged. The network layer is used to indicate how valuable the link is. If the pin bit is set, then the neighbor entry cannot be removed. If the the compare bit is set, then the metric value is better.

3-4 Work Flow of SD-WSN

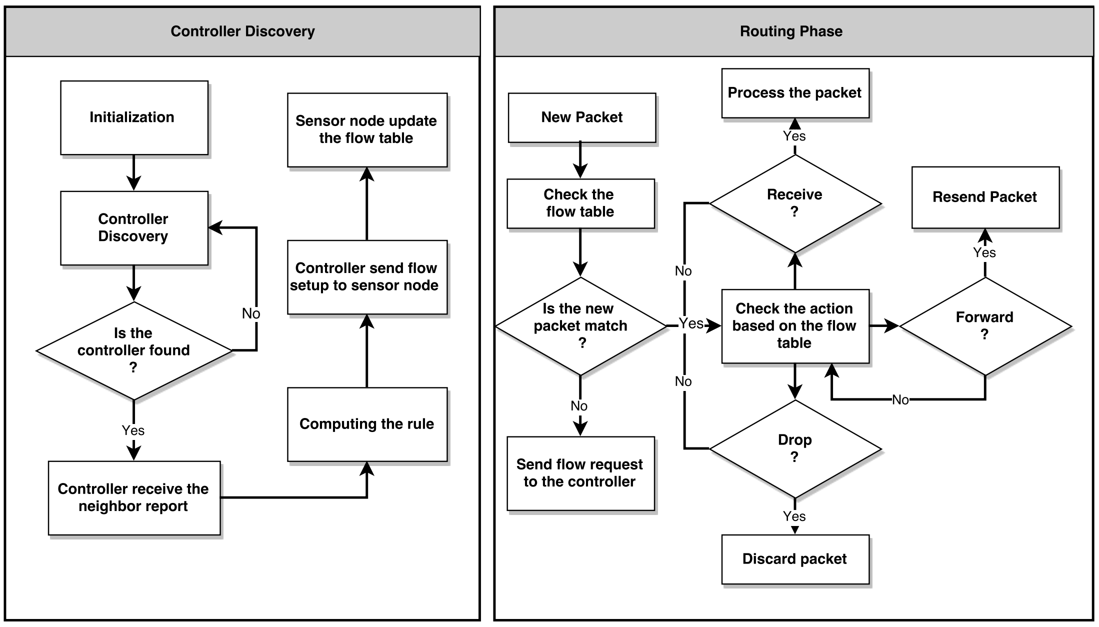

Controller Discovery
Initialization
Controller Discovery
Is the controller found ?
No
Yes
Controller receive the neighbor report
Computing the rule
Controller send flow setup to sensor node
Sensor node update the flow table
Routing Phase
New Packet
Check the flow table
Is the new packet match ?
No
Send flow request to the controller
Yes
Check the action based on the flow table
Receive ?
Yes
Process the packet
No
Drop ?
Yes
Discard packet
Forward ?
Yes
Resend Packet
No

Figure 3-7: Controller Discovery and Routing Phase

3-4-1 Neighbor Discovery/Controller Discovery

Initially, when the *SDN sensor node* is deployed, each *SDN sensor node* does not know who is the *SDN controller node* and the number of its neighboring sensor nodes. As soon as an *SDN sensor node* boots up, the first step for each *SDN sensor node* is performing neighbor discovery and controller discovery procedure by broadcasting a beacon packet to all its neighbors. The structure of the beacon packet is shown by Listing 3.2.

```
typedef nx_struct rssi_beacon_msg{
  nx_uint16_t source_id;
  nx_uint16_t controller_id;
  nx_uint16_t metric;
  nx_bool findController;
}rssi_beacon_msg;
```

Listing 3.2: Beacon Packet Struct

In the process of neighbor discovery, *SDN sensor nodes* employs CTP. Upon reception of a beacon packet, the sensor node which receives the broadcast packet forwards an acknowledgment (ACK) packet to the original sensor node. When an ACK is received by a sensor node, the sensor node extracts all the relevant information about its neighbors, such as the neighbor ID and the ETX link value. This information is added to a list of neighbor sensor nodes. The example of the neighbor table shown in Table 3-2 and the structure of neighbor table are defined by Listing 3.3 and Listing 3.4.

```
typedef nx_struct {
  nx_am_addr_t neighbor_id;
  nx_uint16_t etx;
  nx_uint16_t rssi;
} nx_neighbor_table_entry;
```

Listing 3.3: Neighbor Table Entry Struct

```
typedef nx_struct {
  nx_uint8_t numOfNeighbors;
  nx_neighbor_table_entry neighbors[
      NEIGHBOR_TABLE_SIZE];
} nx_neighbor_table;
```

Listing 3.4: Neighbor Table Struct

```
typedef struct {
  nx_uint16_t origin;
} tinysdn_set_controller_t;
```

Listing 3.5: Set Controller Struct

The *SDN controller node* is predefined by ETX value 0 and the structure of defining the controller is denoted by Listing 3.5 . When the neighbor report reaches the *SDN controller node*, the *SDN controller node* extracts the report, then the *SDN controller node* can use the information from the neighbor report to make the rule (flow table) for each of the *SDN sensor nodes*. The concept of flow table is similar to OpenFlow [26] which defines the action for each *SDN sensor node*. When the *SDN controller node* computes the flow table for each *SDN sensor node*, the *SDN controller node* broadcasts the flow table to each *SDN sensor node*. Upon reception of the flow table, the *SDN sensor nodes* recognize that the controller discovery is done and the *SDN sensor node* will update the flow table. Figure 3-8 shows the implementation of data collection in SD-WSN.

Table 3-2: Neighbor Table

Neighbor ID	Link Quality
2	10
3	20
4	30

3-4-2 Routing Phase

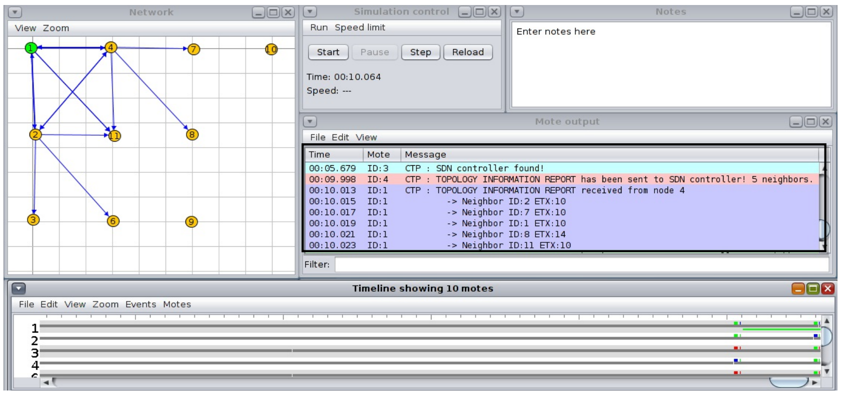
The routing phase is started by the reception of the flow table from the *SDN controller node*. When a *SDN sensor node* receives the flow table, the *SDN sensor node* stops controller discovery. The flow table defines the next hop for the *SDN sensor node*. The *SDN sensor node* then forwards the packet according to the next hop defined by the flow table. When a *SDN sensor node* receives a packet during the routing phase, the sensor node will check the packet in the flow table. If the new packet is not defined, the *SDN sensor node* will send flow request to the *SDN controller node*. If the packet is matched with the flow table, the *SDN sensor node* will execute the action based on the flow table rule.

```
typedef nx_struct
    tinysdn_flow_request_ctp {
  nx_uint8_t type;
  nx_uint16_t origin;
  nx_uint16_t target;
} tinysdn_flow_request_ctp_t;
```

Listing 3.6: Flow Request Struct

Listing 3.6 denotes the structure of flow request, which consists of three parameters: type, origin, and target. Type, to specify the request of the flow; origin, to show the originating address of the packet; and target, to define the flow which is used for the packets.

The flow table defines the action for each *SDN sensor node*. The three actions defined by the *SDN controller node* are: Drop, receive, and forward. In drop action, the packet will be discarded since the packet is not intended for that particular *SDN sensor node*. In receive action, the packet will be accepted, while forward action, the packet is transmitted to the next hop defined by its flow table.



```
Time        Mote   Message
00:05.679   ID:3   CTP : SDN controller found!
00:09.998   ID:4   CTP : TOPOLOGY INFORMATION REPORT has been sent to SDN controller! 5 neighbors.
00:10.013   ID:1   CTP : TOPOLOGY INFORMATION REPORT received from node 4
00:10.015   ID:1         -> Neighbor ID:2 ETX:10
00:10.017   ID:1         -> Neighbor ID:7 ETX:10
00:10.019   ID:1         -> Neighbor ID:1 ETX:10
00:10.021   ID:1         -> Neighbor ID:8 ETX:14
00:10.023   ID:1         -> Neighbor ID:11 ETX:10
```

Figure 3-8: The Implementation of Data Collection in SD-WSN

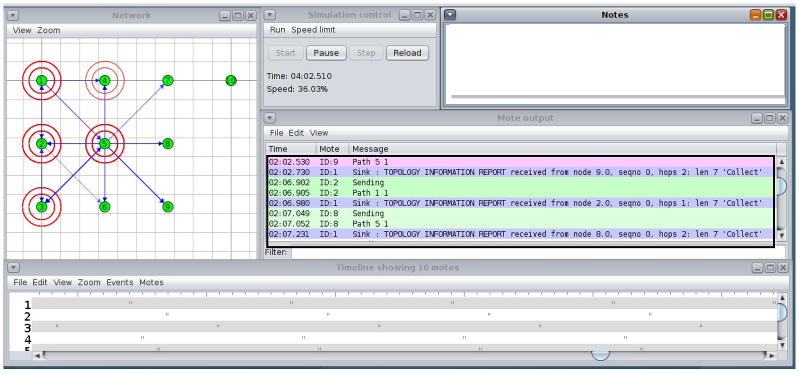
3-5 Work Flow of WSN

The implementation of WSN work flow is based on the Rime data collection protocol using the Contiki operating system [10]. The explanation of WSN work flow using Rime data collection protocol consists of two steps: setting up the collection tree and sending messages towards the sink. The sensor nodes use *identified polite broadcast primitives (ipolite)* channel for creating the collection tree. After the tree has been created, the sensor nodes send the message by using *hop-by-hop reliable unicast channel (ruc).*

In the process of setting up the collection tree, the sensor nodes send a periodic announcement containing the number of hops from the sensor nodes to the sink through *ipolite*. The *ipolite* channel stores the list of packet attributes in queue buffer and sets up a timer (listen period and transmission period). During the listen period, the sender node listens for other transmission. If it hears a packet that matches the attributes in queue buffer, the sender node drops the packet. Otherwise, the sender node broadcasts its packet to all local neighbors during transmission period. In other words, *ipolite* is used to avoid multiple copies of messages.

In the process of sending the message, *ruc* uses acknowledgements and retransmissions to ensure the local neighbors receive the packet. If the receiver node has acknowledged the packet, the *ruc* notifies the sending application. Otherwise, *ruc* retransmits the acknowledgement packet.

The difference between SD-WSN and WSN lies in the mechanism for choosing the path from the source to the sink. In SD-WSN, the route for *SDN sensor node* is determined by the *SDN controller node*, whereas in WSN, the route from sensor node to the sink is determined by each sensor node. The collection protocol in WSN is an address-free protocol, and thus the sensor nodes send packets toward a sink node without predefined routes, as opposed to the concept of a collection protocol on SD-WSN where the path for each *SDN sensor node* is determined by the *SDN controller node*.



Time: 04:02.510
Speed: 36.03%

```
Time       Mote  Message
02:02.530  ID:9  Path 5 1
02:02.730  ID:1  Sink : TOPOLOGY INFORMATION REPORT received from node 9.0, seqno 0, hops 2: len 7 'Collect'
02:06.902  ID:2  Sending
02:06.905  ID:2  Path 1 1
02:06.980  ID:1  Sink : TOPOLOGY INFORMATION REPORT received from node 2.0, seqno 0, hops 1: len 7 'Collect'
02:07.049  ID:8  Sending
02:07.052  ID:8  Path 5 1
02:07.231  ID:1  Sink : TOPOLOGY INFORMATION REPORT received from node 8.0, seqno 0, hops 2: len 7 'Collect'
```

Figure 3-9: The Implementation of Data Collection in WSN

The controllers on the WSN are coupled and distributed on each sensor node, thus each node sensor can select the paths automatically based on the routing protocol used. The protocol in WSN builds a tree structure from the sink node by sending periodic announcements which contain the number of hops from each sensor node to the sink node. After the process of building the tree has finished, the sensor nodes start sending packets toward the sink. Figure 3-9 shows the implementation of data collection in WSN.

3-6 Scenario

Network performance can vary due to the WSN device's capabilities or the network environments [2]. To simulate the performance of SD-WSN, some network parameters are changed to measure the network performance over a range of environments. Three types of network environments were used to test this network: homogeneous network, heterogeneous network, and dynamic network. The following section explains these parameters.

3-6-1 Network Topology

Table 3-3: Network Topology

5 Motes		**15 Motes**	
Source (Node Id)	**Neighbors (Node Id)**	**Source (Node Id)**	**Neighbors (Node Id)**
1	2, 3	1	2, 4, 5
2	1, 4	2	1, 3, 4, 5, 6
3	1, 4, 5	3	2, 5, 6
4	2, 3	4	1, 2, 5, 7, 8
5	3	5	1, 2, 3, 4, 6, 7, 8, 9
10 Motes		6	2, 3, 5, 8, 9
Source (Node Id)	**Neighbors (Node Id)**	7	4, 5, 8, 10, 11
1	2, 4, 5	8	4, 5, 6, 7, 9
2	1, 3, 4, 5, 6		10, 11, 12
3	2, 5, 6	9	5, 6, 8, 11, 12
4	1, 2, 5, 7, 8	10	7, 8, 11, 13, 14
5	1, 2, 3, 4, 6, 7, 8, 9	11	7, 8, 9, 10
6	2, 3, 5, 8, 9		12, 13, 14, 15
7	4, 5, 8, 10	12	8, 9, 11, 14, 15
8	4, 5, 6, 7, 9, 10	13	10, 11, 14
9	5, 6, 8	14	10, 11, 12, 13, 15
10	7, 8	15	11, 12, 14

There are two types of network topology that will be compared: SD-WSN and the existing WSN (Rime data collection protocol, Contiki). Each network topology consists of 5, 10, and 15 motes (sensor nodes, *SDN sensor nodes*, *SDN controller node*). The illustration is shown in Figure 3-10.

In SD-WSN, sensor node number 1 (blue mote) acts as the *SDN controller node* and the other *SDN sensor nodes* transmit packets to the *SDN controller node*. On the other hand, in WSN the sensor node number 1 (blue mote) acts as the sink and the other sensor nodes transmit packets to the sink.

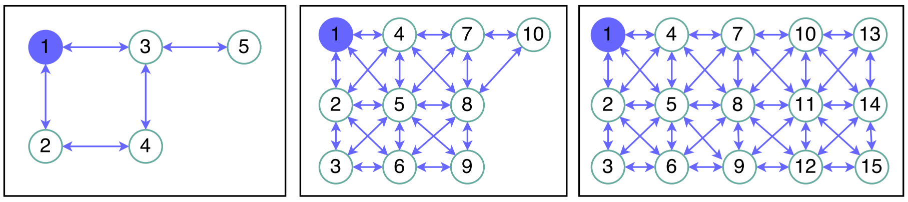


Figure 3-10: The Illustration of Network Topology

3-6-2 Simulation Setup

To perform the simulation and performance evaluation of SD-WSN, several parameters are divided into three parts, namely general parameters, homogeneous network parameters, and heterogeneous network parameters. Homogeneous network and heterogeneous network are used to define the capability of motes (sensor nodes, *SDN sensor nodes*, *SDN controller node*) in terms of the transmission range and sensing range [46]. Table 3-4 summarizes the simulation setup in our thesis.

Table 3-4: Simulation Setup

No	Parameters	Details
1	Node placement	Fixed
2	The number of motes	5, 10, 15
3	Routing protocol	CTP (SD-WSN), Rime Data Collection, Contiki (WSN)
4	Simulation time	12 Minutes
5	Network circumstances	Normal and Dynamic
6	Topology	Grid
7	Sampling Period	30s
Homogeneous Network		
1	Radio Medium	UDGM
2	Tx Range	50m
3	INT Range	100m
4	Tx and Rx Ratio	100%
Heterogeneous Network		
1	Radio Medium	DGRM
2	Rx Ratio	20%-100%
3	RSSI	((-10) - (-18)) dBm
4	LQI	(105 - 65)

3-6-3 Homogeneous Network

A homogeneous network is a network comprised of motes using similar configurations. In our simulation, all the motes are identical in terms of transmission range (Tx range: The range in which the transmitted packet within this range can be received correctly by any mote), interference range (INT range: The range in which the packet transmission can be heard by any mote within this range, but the packet transmission cannot be received correctly), and the successful transmission-reception of the radio packet (Tx and Rx Ratio: The packet from each mote can be sent and received with a certain probability of success). Figure 3-11 illustrates the concept a homogeneous network.

To simulate the homogeneous network (a condition where all the motes have the same range and success ratio), the radio medium is designed, using Unit Disk Graph Medium (UDGM) Constant loss. The radio medium is modelled with a transmission range of 50m, interference range of 100m, and transmission success ratio and receives success ratio of 100%. Figure 3-15 shows the implementation of homogeneous network configuration.

3-6-4 Heterogeneous Network

In a heterogeneous sensor network, two or more motes are designed with different link qualities, and success ratio, but the number of motes with fixed node placement in the heterogeneous network are similar to the homogeneous network. Figure 3-12 illustrates the concept of heterogeneous network.

Table 3-5: The Initial Configuration for Each Motes in Heterogeneous Network

RX Ratio	RSSI (dBm)	LQI
100%	-10	105

A different link quality is used, and this makes the network more complex than the homogeneous network. To create a heterogeneous network, directed graph radio medium (DGRM) is used. With DGRM, several parameters are varied, such as reception ratio, received signal strength indicator (RSSI), and link quality indicator (LQI). The reception ratio is set from 20% to 100%, the received signal strength indicator (RSSI) is set from -10 dBm to -18 dBm, and the link quality indicator (LQI) is set from 105 to 65. RSSI denotes an estimation of the average signal power received, while LQI values denotes the quality of the received packet. higher values of RSSI and LQI indicate higher quality. [5].

Table 3-5 shows the initial configurations for each mote in heterogeneous network. Some motes which have a high degree of neighborhood are given different configurations. These motes configuration (Table 3-6) are used to represent heterogeneous networks. Figure 3-15 shows the implementation of heterogeneous network configuration.

3-6-5 Dynamic Network

To measure how effectively the flow table can adapt to *SDN sensor node* failures, a network is designed which contains joining and leaving sensor nodes. After four minutes, we remove several motes that have a high degree of neighborhood and after four minutes those sensor nodes rejoin the network. When one of these motes is off, routes which currently flow through its mote will fail. This condition triggers the route discovery phase to reconnect *the SDN controller node* or sink. Figure 3-16 denotes the gray circles as the dynamic motes.

The parameters for the dynamic network is similar to the homogeneous network where the radio medium is UDGM, Tx range is 50m, INT range is 100m, and the Tx-Rx ratio is 100%.

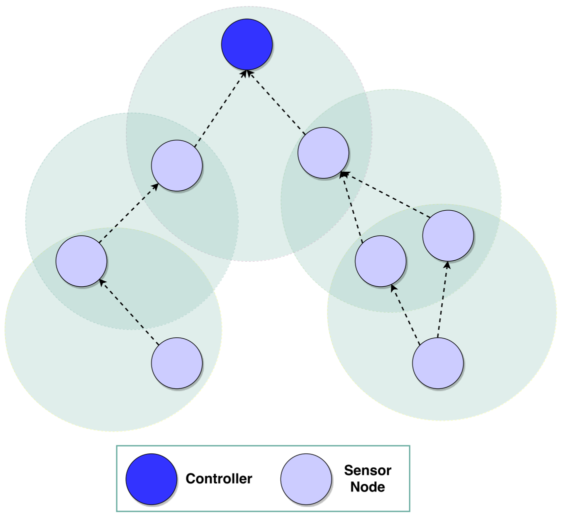


Figure 3-11: A Homogeneous Network

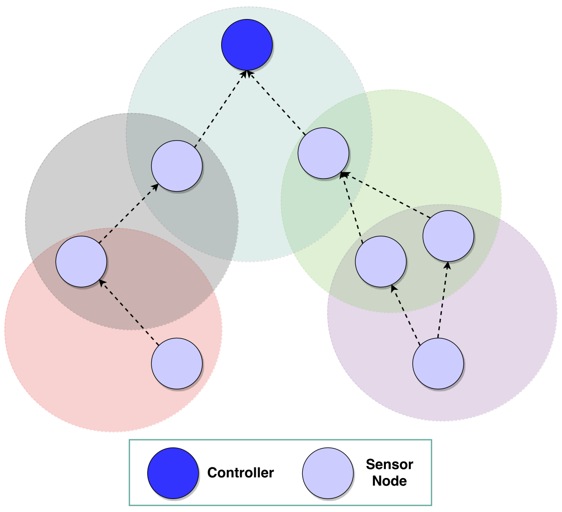


Figure 3-12: A Heterogeneous Network

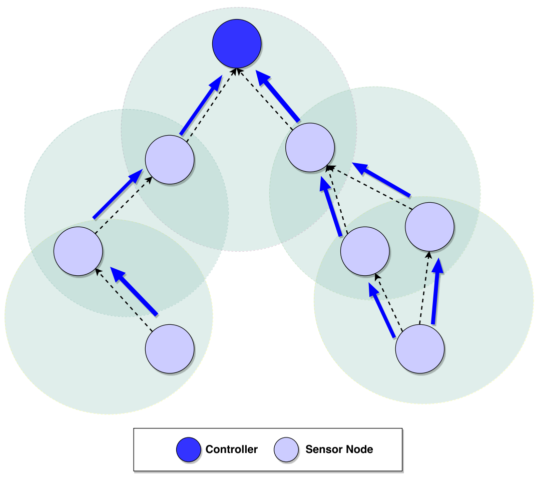


Figure 3-13: A stable network

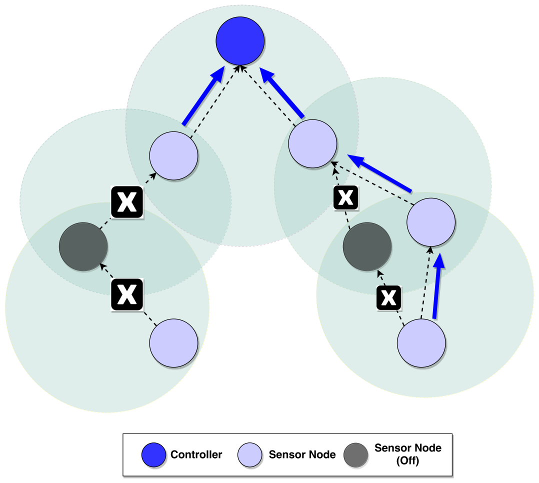


Figure 3-14: A dynamic network

Table 3-6: The Configuration of Heterogeneous Networks for Several Motes

5 Motes			
Node ID	**RX Ratio**	**RSSI (dBm)**	**LQI**
3 <->1	90%	-11	100
3 <->4	80%	-12	95
3 <->5	70%	-13	90
10 Motes			
Node ID	**RX Ratio**	**RSSI (dBm)**	**LQI**
5 <->1	90%	-11	100
5 <->2	80%	-12	95
5 <->3	70%	-13	90
5 <->4	60%	-14	85
5 <->6	50%	-15	80
5 <->7	40%	-16	75
5 <->8	30%	-17	70
5 <->9	20%	-18	65
15 Motes			
Node ID	**RX Ratio**	**RSSI (dBm)**	**LQI**
5 <->1, 11 <->7	90%	-11	100
5 <->2, 11 <->8	80%	-12	95
5 <->3, 11 <->9	70%	-13	90
5 <->4, 11 <->10	60%	-14	85
5 <->6, 11 <->12	50%	-15	80
5 <->7, 11 <->13	40%	-16	75
5 <->8, 11 <->14	30%	-17	70
5 <->9, 11 <->15	20%	-18	65

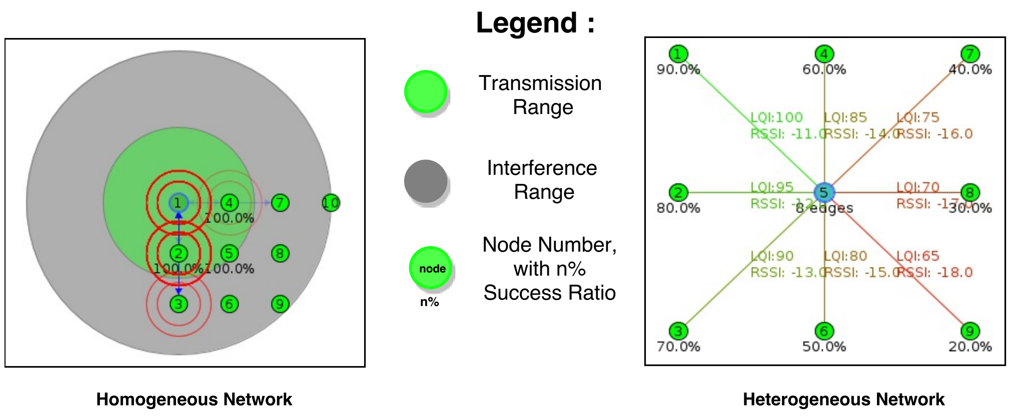


Figure 3-15: The Implementation of Homogeneous and Heterogeneous Network

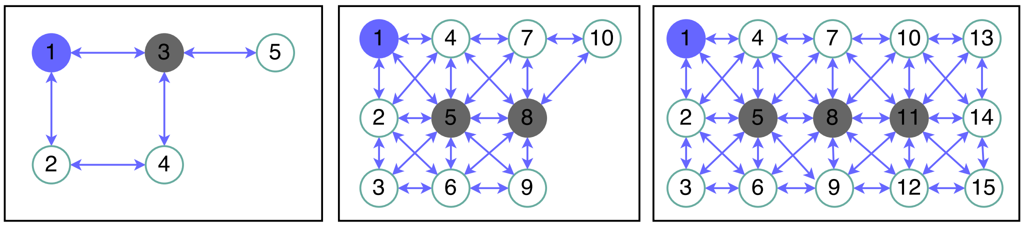


Figure 3-16: Dynamic Motes

Figure 3-13 denotes the normal network where the motes forward a packet to the controller and Figure 3-14 illustrates the dynamic network where two motes in the stable network are off.

3-7 Performance Metrics

- **Duty Cycle.**

  Power consumption is important for the motes to achieve a long network lifetime. For instance, the CC2420 radio transceiver, used in the TelosB, draws approximately 60 milliwatts of power when it is listening for radio traffic, and then its power consumption is slightly higher when transmitting radio data. With a power draw of 60 milliwatts, a mote depletes its batteries in a matter of days [8]. The most widely used approach to calculate energy consumption for each mote node is to monitor the duty cycle of the radio [24]. Duty cycle denotes the proportion of the time a mote in the active state (t_{active}) over the total time (t_{total}) [37].

$$Duty\ Cycle = (\frac{t_{active}}{t_{total}})100 \tag{3-5}$$

  In our simulation, the duty cycle is a combination between Radio Tx time and Rx time and it is expressed as a percentage. Radio Tx time and Radio Rx time denote how long the motes transmit and receive packets, respectively. By investigating the percentage of time radio is being kept active, i.e. by keeping track of the duty cycle for each mote, we can estimate the network lifetime. The higher the duty cycle means the shorter the network lifetime and the lower duty cycle represents more energy will be saved by the motes. In other words, a low duty cycle mote has a much longer lifetime [43].

- **Delay**

  WSN is used for many applications such as health monitoring and surveillance system. Those WSN applications require QoS guarantees to fulfill the requirements, for instance, real-time data delivery. According to this requirement, delay is one of the major factors which affect QoS [44]. In this simulation, delay is defined as the time difference between when a packet is sent by a mote and the time a packet is received by the *SDN controller node* (SD-WSN) or the sink (WSN). If we let ($t_{\ tx,mote}$) be the time a mote sends the packet and ($t_{\ rx,controller/sink}$) be the time when a *SDN controller node*/sink receives the packet.

$$Delay = t_{tx,mote} - t_{rx,controller/sink} \tag{3-6}$$

- **Packet Delivery Ratio**

  Packet Delivery Ratio (PDR) is the ratio between the number of received packets by the *SDN controller node* (SD-WSN) or the sink (WSN) ($msg_{\ rx,\ controller/sink}$) and the number of packets transmitted by a mote ($msg_{\ tx,\ mote}$). If the PDR is equals to one, it can be said that all the packets have been received successfully by the *SDN controller node*/sink. In the worst case, none of the packets reaches the *SDN controller node*/sink. This may happen if the controller is disconnected from the network and causes the packet delivery ratio to be zero [27]

$$PDR = \frac{msg_{rx,controller/sink}}{msg_{tx,mote}} \tag{3-7}$$

- **Packet Duplication**

  Packet duplication is a condition when a single packet may be forwarded by multiple neighbors simultaneously, which can increase the channel occupancy and the energy consumption in the network [36]. This problem can be caused by several factors such as routing loop on a path, lost acknowledgments, packet re-transmissions and packet merging [17]. SD-WSN and WSN use multiple hops to send a packet from the motes to the *SDN controller node*/sink. If each hop generates one duplicate packet, on the next hop, there will be multiple duplicate packets. Over a small-scale network, this is not a crucial issue. However, in the large-scale network which has many hops from the motes to the *SDN controller node*/sink, this is an important issue as the packet duplication can increase significantly. The ratio of packet duplication can be obtained as follows:

$$Packet\ Duplication = \frac{(msg_{rx,controller/sink} - msg_{tx,mote})}{msg_{tx,mote}} \tag{3-8}$$

  where $msg_{\ rx,controller/sink}$ is the number of received packets by the *SDN controller node* (SD-WSN) or the sink (WSN) and $msg_{\ tx,\ mote}$ is the number of packets transmitted by the mote.

Chapter 4

Experimental Results and Analysis

This chapter discusses the simulation of the proposed SD-WSN and its performance against an existing WSN, where the simulations were conducted on an ubuntu 14.04 LTS, Intel (R) Core i7-4510U 2.00 GHz with 4 GB RAM memory. The implementation is evaluated based on four performance metrics: Delay, packet delivery ratio, duty cycle, and packet duplication with three network conditions: Homogeneous network, heterogeneous network, and dynamic network.

4-1 CTP Verification on SD-WSN

To determine the extent to which CTP on SD-WSN can produce a reliable ETX link value, a comparison of ETX link value between SD-WSN and WSN are performed by using collect protocol through the Contiki shell (CollectView). The verification process is performed during the neighbor/controller discovery phase, when the *SDN controller node* receives the neighbor report of each *SDN sensor node*. The verification process uses three types of scenarios: Homogeneous network, heterogeneous network, and dynamic network, while the number of motes are 5, 10, and 15.

Figure 4-1 shows the tree structure of ETX link value on WSN, and the comparison results of ETX link values on SD-WSN and WSN are shown by Figure 4-2 and Figure 4-3. Figure 4-2 shows that the ETX link values are relatively similar for SD-WSN and WSN in the homogeneous and heterogeneous network. However, as the number of nodes is increased, the ETX link values for some nodes in SD-WSN are slightly different from the ETX value in WSN.

Figure 4-3 shows the ETX link values in the dynamic network which are divided into three parts: 0-4 minutes, all the motes are on; 4-8 minutes, some of the motes are off; and 8-12 minutes, all the motes are on again. When the number of motes is five, the ETX link values for SD-WSN and WSN are equal, but when the number of motes is increasing from 5 to 15, the ETX link values for some *SDN sensor nodes* in SD-WSN are different from WSN. A significant difference of ETX values is observed when the number of the *SDN sensor nodes* is 15 in the dynamic network. Since the determination of the SD-WSN path is defined by the *SDN controller node*, when some *SDN sensor nodes* are off, the *SDN sensor nodes* must perform routing discovery to update the ETX link value. Consequently, SD-WSN slowly detects the failures and takes longer adaptation to find the best path for each *SDN sensor node* because the path for each *SDN sensor node* is specified by the *SDN controller node*.

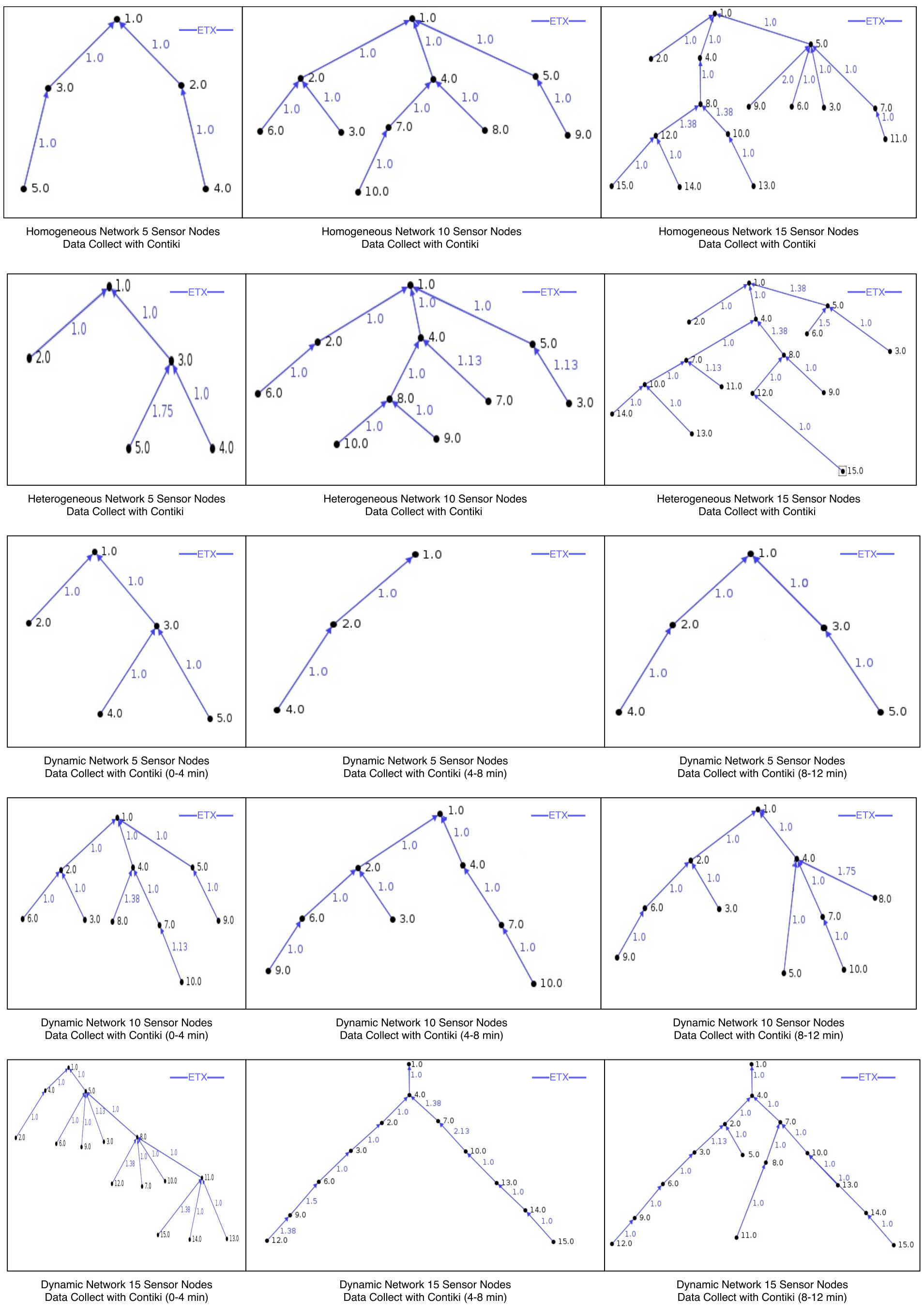

ETX
Homogeneous Network 5 Sensor Nodes Data Collect with Contiki
Homogeneous Network 10 Sensor Nodes Data Collect with Contiki
Homogeneous Network 15 Sensor Nodes Data Collect with Contiki
Heterogeneous Network 5 Sensor Nodes Data Collect with Contiki
Heterogeneous Network 10 Sensor Nodes Data Collect with Contiki
Heterogeneous Network 15 Sensor Nodes Data Collect with Contiki
Dynamic Network 5 Sensor Nodes Data Collect with Contiki (0-4 min)
Dynamic Network 5 Sensor Nodes Data Collect with Contiki (4-8 min)
Dynamic Network 5 Sensor Nodes Data Collect with Contiki (8-12 min)
Dynamic Network 10 Sensor Nodes Data Collect with Contiki (0-4 min)
Dynamic Network 10 Sensor Nodes Data Collect with Contiki (4-8 min)
Dynamic Network 10 Sensor Nodes Data Collect with Contiki (8-12 min)
Dynamic Network 15 Sensor Nodes Data Collect with Contiki (0-4 min)
Dynamic Network 15 Sensor Nodes Data Collect with Contiki (4-8 min)
Dynamic Network 15 Sensor Nodes Data Collect with Contiki (8-12 min)

Figure 4-1: ETX Graph of WSN

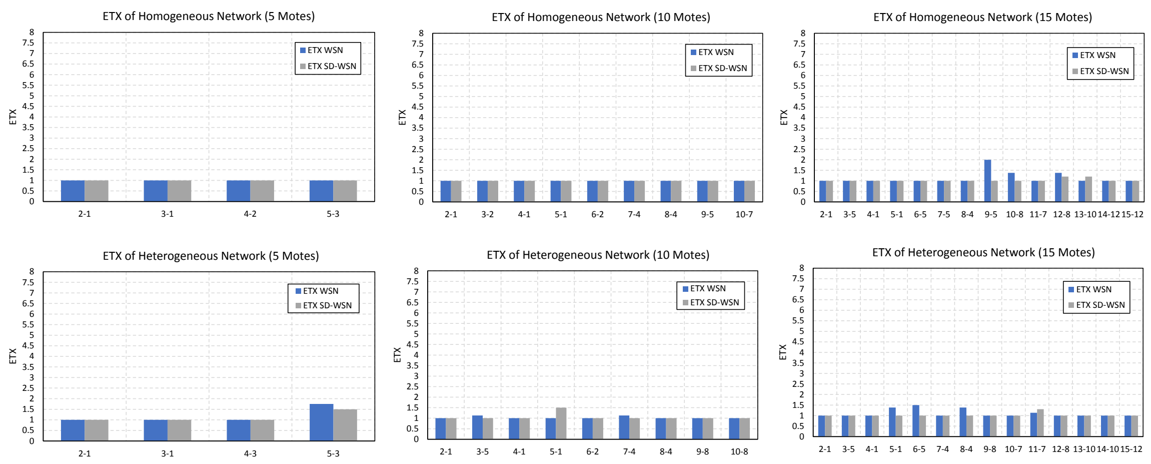


Figure 4-2: ETX of SD-WSN and WSN
(Homogeneous and Heterogeneous Network)

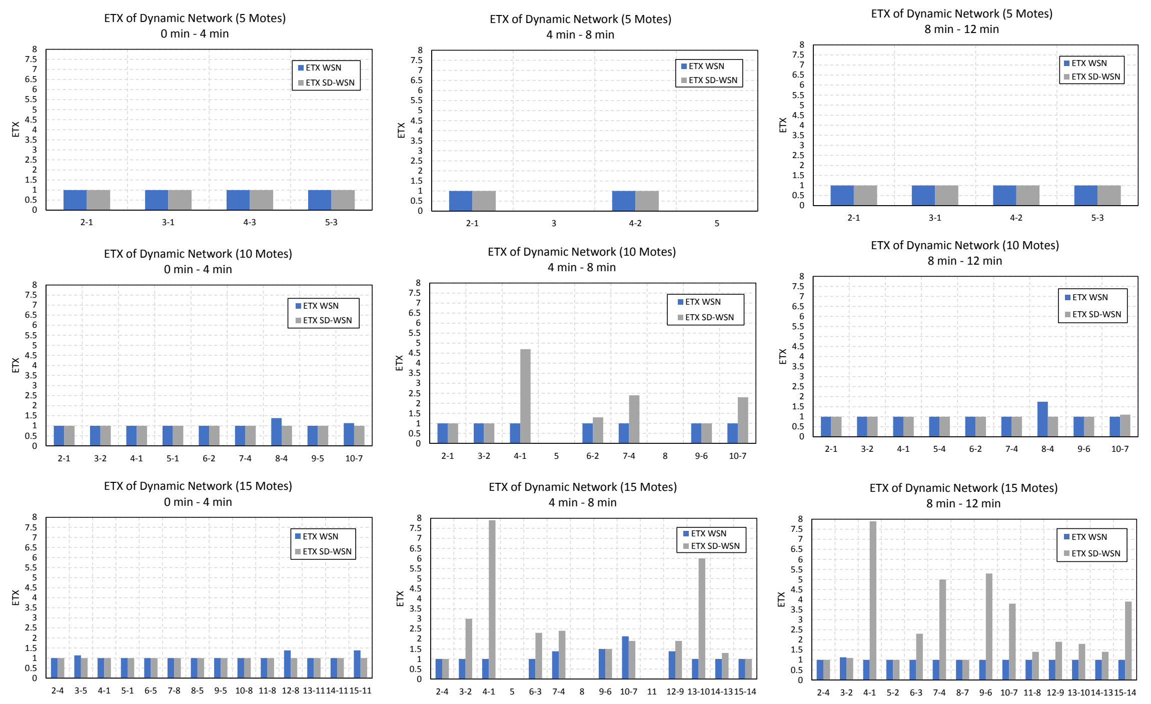


Figure 4-3: ETX of SD-WSN and WSN
(Dynamic Network)

4-2 Average Delay

Figure 4-4 shows the average delay of WSN and SD-WSN in three scenarios: Homogeneous network, heterogeneous network, and dynamic network. In general, increasing the number of motes from 5 to 15 will increase the average delay gradually, both for WSN and SD-WSN. The increasing ratio of average delay in homogeneous network and heterogeneous network for

WSN and SD-WSN is relatively similar, where the ratio of average delay ranges from 1.2 to 1.9. However, in the dynamic network, the ratio of average delay for SD-WSN rises from 2.03 (when the number of *SDN sensor node* grows from 5 to 10) to 2.3 (when the number of *SDN sensor node* grows from 10 to 15). On the other hand, the ratio of average delay for WSN is only around 1.6 (when the number of sensor nodes grows from 5 to 10) to 1.98 (when the number of sensor nodes grows from 10 to 15). The significant increase of SD-WSN average delay in dynamic network indicates that the use of *SDN controller node* causes the *SDN sensor node* to be less able to adapt to dynamic network changes so that each *SDN sensor node* takes a long time when some *SDN sensor node* are leaving and joining the network. The performance deficit in terms of the average delay can also be attributed to the increased overhead caused by maintaining the entire *SDN sensor node* by the *SDN controller node.*

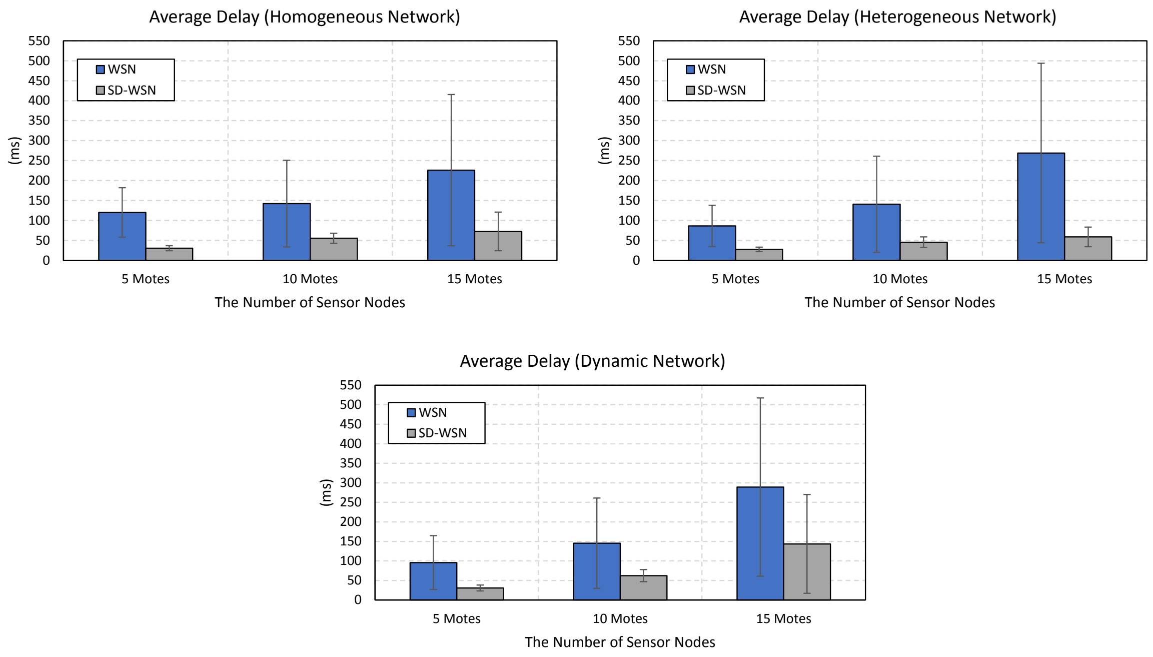


Figure 4-4: Average Delay

Increasing the number of motes also affects the standard deviation. Because a longer delay is to be expected for further distance between the motes and the *SDN controller node*/sink. Figure 4-4 denotes the standard deviation of WSN is higher than the standard deviation of SD-WSN as WSN uses identified polite broadcast primitive (*ipolite*) during the process of setting up the collection tree. In this mechanism, *ipolite* stores the outgoing message in queue buffer. Then, it sets up a random timer during the second half of the interval time. The timeline algorithm of polite broadcast primitive is shown in Figure 4-5. If a packet of outgoing message with the same header is received from its neighbor within the interval, the packet is not sent. This mechanism aims to reduce the total amount of packet transmission by reducing the messages that other sensor nodes have already sent [10]. However, using this mechanism in WSN can increase the average delay and standard deviation since Rime Data Collection protocol uses random transmission period for setting up the collection tree.

Figure 4-4 shows the average delay of SD-WSN is lower around 34% than the average delay

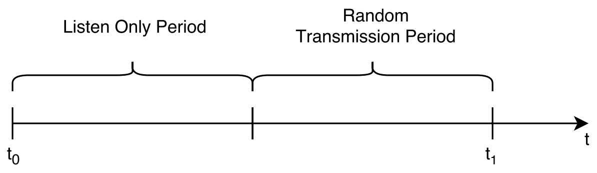


Figure 4-5: The Timeline of Polite Broadcast Primitive Algorithm [10]

of WSN as SD-WSN uses the *SDN controller node* in which the routing of sending a message is defined by the *SDN controller node*. Thus, the time required by each *SDN sensor node* to send a packet is shorter than WSN.

4-3 Packet Delivery Ratio (PDR)

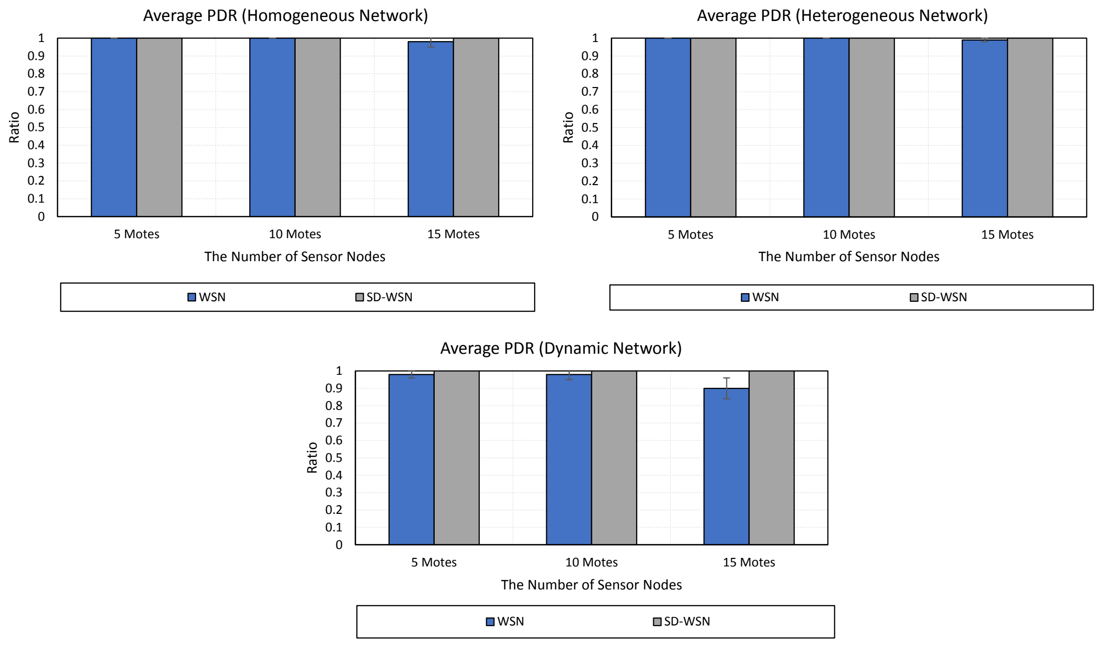


Figure 4-6: Average Packet Delivery Ratio

Figure 4-6 denotes the performance metric of WSN and SD-WSN from the perspective of packet delivery ratio in homogeneous network, heterogeneous network, and dynamic network. In homogeneous and heterogeneous network, we observed that with the increasing number of the motes from 5 to 15, the average PDR for WSN and SD-WSN are relatively similar (around 0.98 to 1). However, in dynamic network, the increment of the motes from 5 to 15 affects the performance of WSN: the average PDR of WSN in a dynamic network decreases from 0.98 to 0.9, whereas the average PDR of SD-WSN is 1. The packets lost during the process of delivery messages from the sensor node to the controller/sink indicates that there are inaccuracies in the link estimation.

According to author's investigation, one of the causes of packet delivery ratio value for SD-WSN always 1 in the three scenario is due to packet duplication. On the one hand, packets duplication can increase the packet delivery ratio because their redundancy can prevent packet drops [17], but on the other hand, packets duplication will increase the load of the network as many duplicate packets are received by the *SDN controller node*.

4-4 Packet Duplication

Figure 4-7 shows the performance metric of WSN and SD-WSN from the perspective of packet duplication in a homogeneous network, heterogeneous network, and dynamic network. This performance metric is measured by the percentage number of packet duplication at the *SDN controller node* or sink per transmission.

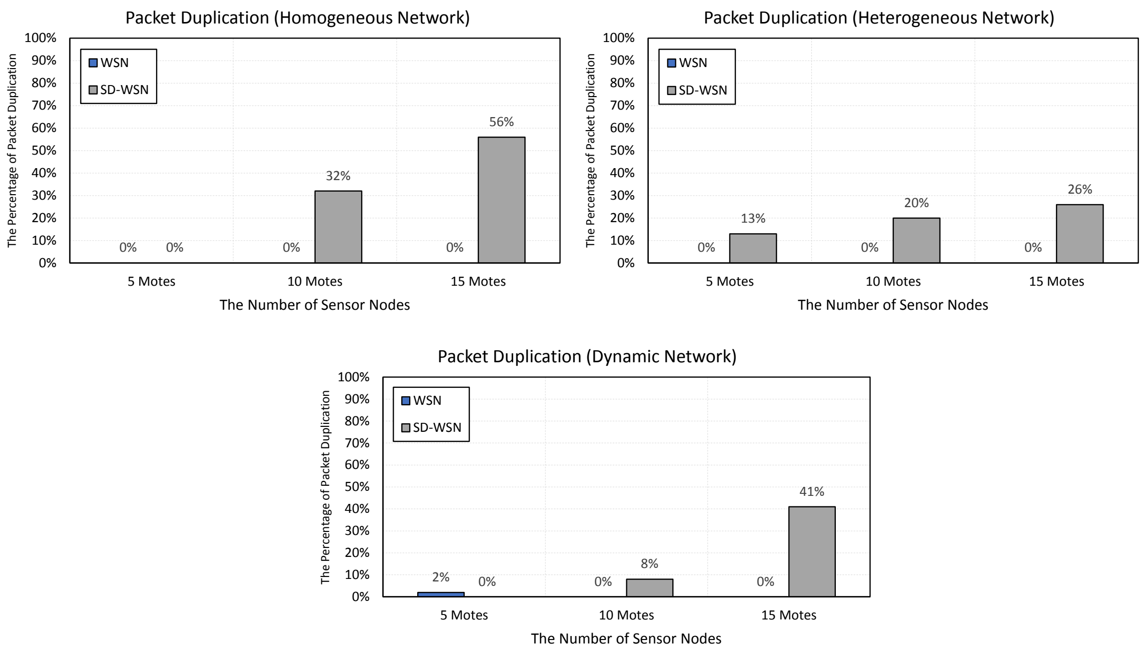


Figure 4-7: Packet Duplication

On average, packets duplication do not occur on WSN (a homogeneous network, heterogeneous network, and dynamic network). Moreover, increasing the number of sensor nodes does not affect the packet duplication on the WSN as well. WSN uses Rime data collection for creating the tree structure and sending messages to the sink. During the neighbor discovery process, Rime data collection uses a single-hop broadcast primitive (*polite*). The polite algorithm is constructed to reduce the total amount of packet transmissions by not repeating messages that other sensor nodes have already sent [10]. Polite broadcast primitive suppresses the multiple copies of the packet by only allowing a single message to be delivered to the sender.

On the other hand, packets duplication occurs in SD-WSN. The results from Figure 4-7 shows that the increasing number of *SDN sensor nodes* have a significant effect on packet duplication in SD-WSN. Increased packet duplication occurs significantly on the dynamic network. When the number of *SDN sensor nodes* is 5 (one *SDN sensor node* leaving and rejoining network), there is no packet duplication. However, when the number of *SDN sensor nodes* increases from 10 to 15 (three *SDN sensor nodes* are leaving and rejoining the network), the duplicate packet rises drastically by 33%. In homogeneous network and heterogeneous network of SD-WSN, there is also an increase of packet duplication, when adding *SDN sensor nodes* from 10 to 15, although not as significant as on dynamic networks. The increase of duplication packets on the homogeneous network and the heterogeneous network are 24% and 6%, respectively.

Packet duplication in SD-WSN indicates that SD-WSN cannot suppress the duplicate packet when an inconsistency occurs. Due to SD-WSN's inability to suppress the packet duplication, increasing the node could easily generate the copies of a packet in the network significantly.

The control plane is responsible for forming a topology and selecting routes, but due to the centralized controller, each *SDN sensor node* is unable to adapt to changing network conditions. Thus, *SDN sensor node* generates copies of a packet during the process of discovering and fixing routing failures.

4-5 Duty Cycle

Figure 4-8 shows that the increasing number of the motes causes the average duty cycle to increase as well. In other words, the growing number of the motes can reduce the network lifetime. According to Figure 4-8, SD-WSN performs better than WSN regarding the efficiency of the average duty cycle (Tx). The centralized controller can reduce the active time of the sensor node to send packets compared to WSN as routing and path calculations in SD-WSN are determined by the *SDN controller node*. Thus, the *SDN sensor node* simply runs the rules that have been made by the *SDN controller node*, whereas in WSN, the path calculation is determined by each sensor node, so the active time of sending packet is longer in order to find the route from the sensor node to the sink.

One of the factors causing the average of Tx Duty cycle on WSN is being higher than the average of Tx Duty Cycle value on SD-WSN because before the message is sent to the neighbors, the outgoing message stored in a queue buffer, and then the upper layer in Rime data collection protocol sets up a timer. The timer is set by Rime data collection protocol to a random time during the second half of the interval time. When the timer fires and the sender has not yet heard a transmission of the same packet attributes from its neighbors, the sender broadcasts the packet to all its neighbors [10].

Although SD-WSN performs better in the sense of average duty cycle (Tx) compared to WSN, the use of centralized controller will increase the Rx duty cycle. Figure 4-8 shows that the average duty cycle (Rx) of SD-WSN is higher than the average duty cycle (Rx) of SDN. This is due to each *SDN sensor node* in SD-WSN needs to communicate with the *SDN controller node*. In addition, packet duplication is one of the main factors causing the average of Rx duty cycle value in SD-WSN is being higher than that of WSN. Packet duplication causes the *SDN sensor node* to receive multiple copies of the packet. This condition makes the *SDN sensor node* Rx duty cycle in SD-WSN increase drastically as increasing the node could easily generate an exponential number of copies of a packet in the network.

The increase in the average of Rx duty cycle in WSN from the five sensor nodes to the 15 sensor nodes on the homogeneous network, heterogeneous network, and dynamic network are

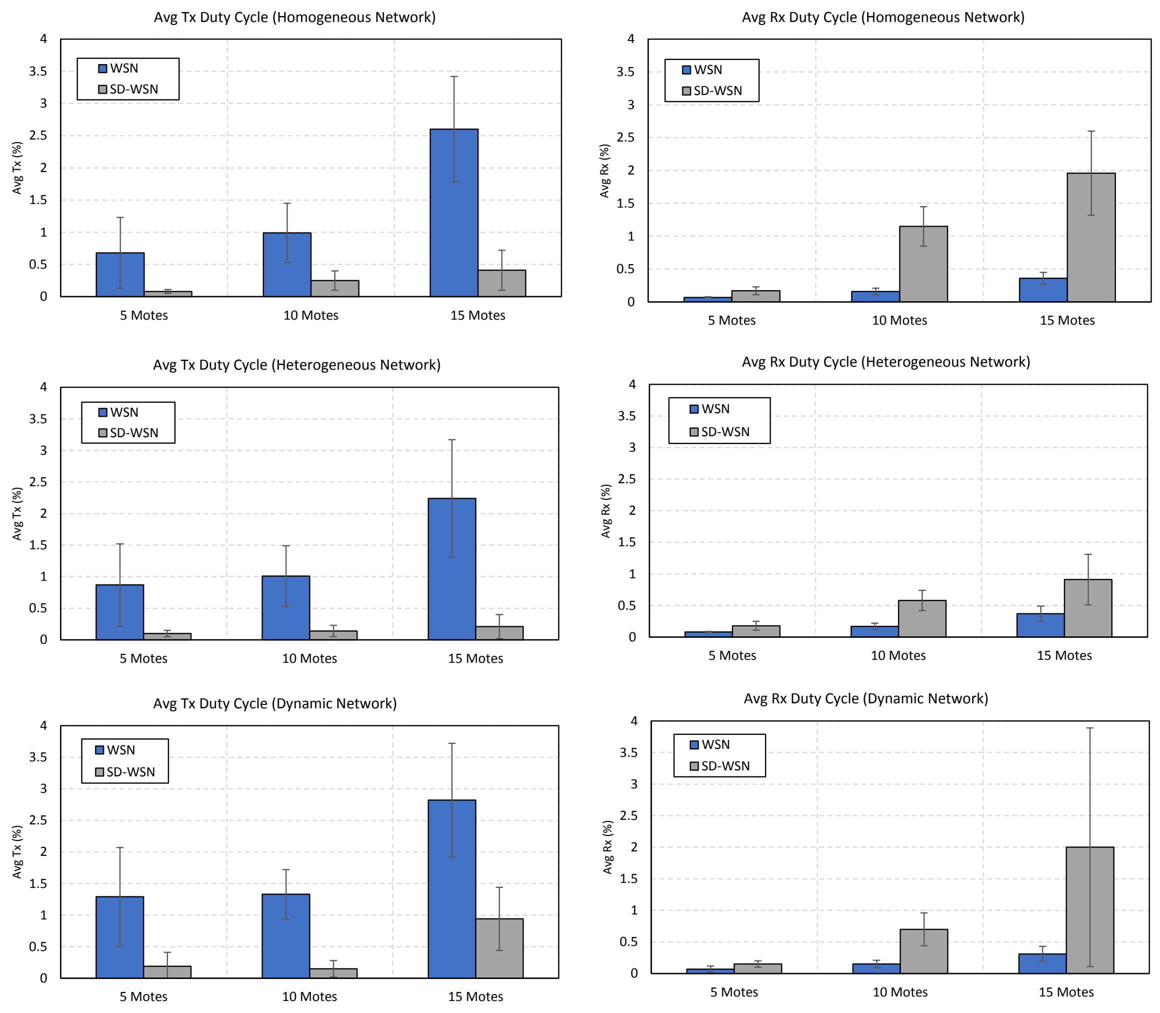


Figure 4-8: Average Duty Cycle (Tx and Rx)

relatively stable at about 0.2%. On the other hand, the increase of the average of Rx duty cycle in SD-WSN from 5 *SDN sensor nodes* to 15 *SDN sensor nodes* on the homogeneous network, heterogeneous network, and dynamic network are quite high at around 1.8%, 0.73%, and 1.9%, respectively.

In a dynamic network environment, the performance of WSN in the sense of Rx duty cycle is more stable than SD-WSN. This is demonstrated by the increasing value of the average and standard deviation of Rx duty cycle which are being relatively similar, around 0.2% and 0.01%, respectively. On the other hand, the values of the average and standard deviation of Rx duty cycle in SD-WSN increase drastically up to 2% and 1.89%, respectively, since each *SDN sensor node* needs to communicate with the *SDN controller node*, so that when the network changes, each *SDN sensor node* cannot automatically make adjustments to the new network conditions.

Figure 4-9 shows the average of Rx duty cycle per sensor node in a dynamic network for SD-WSN and WSN. Although the sensor nodes 5, 8, and 11 are leaving and rejoining the network, there is no significant change in the average of Rx duty cycle on the WSN. On the

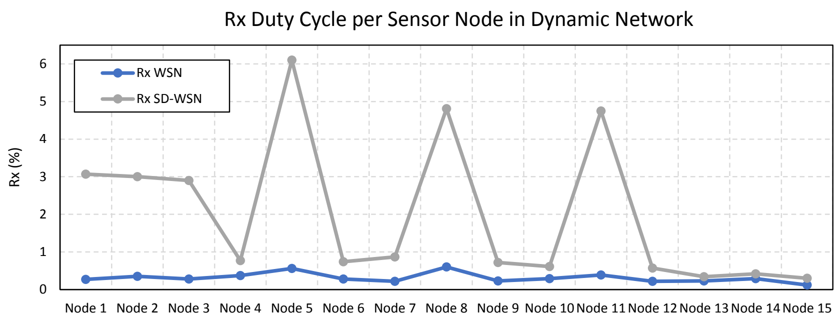


Figure 4-9: Average Duty Cycle per Sensor Node in Dynamic Network

other hand, there is a significant increase in *SDN sensor node* 5, 8, and 11 since these *SDN sensor node* requires a longer time to be able to send or receive packets when leaving and rejoining the network. Figure 4-9 depicts that SD-WSN is not optimally applied to dynamic conditions in the sense of Rx duty cycle because the activity changes from the *SDN sensor node* will affect the network lifetime significantly.

4-6 Clustering Controller

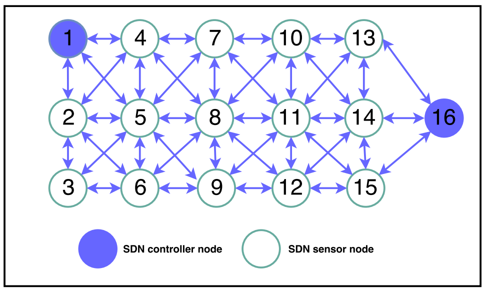


Figure 4-10: Clustering Controller Topology

Based on the results from the previous performance evaluation (Section 4-2, Section 4-3, Section 4-4 and Section 4-5), the increasing number of *SDN sensor node* will decrease the performance of SD-WSN. To solve these issues, we propose using multiple controllers in TinySDN. By using clustering controller in SD-WSN, the multiple controllers is expected to be able to improve the performance of SD-WSN.

The process of finding the multiple controllers in Cluster SD-WSN is similar to finding single *SDN controller node* in SD-WSN. Collection tree protocol is used for creating a tree structure to deliver a message over the network to the *SDN controller nodes*. When multiple *SDN controller nodes* are announced, each *SDN sensor node* joins the lowest ETX value.

Figure 4-10 illustrates the network topology of clustering controller. We use 14 *SDN sensor nodes* and two *SDN controller nodes*. The Cluster SD-WSN is performed in three scenarios: Homogeneous network, heterogeneous network, and dynamic network, while packet delivery ratio, delay, duty cycle, and packet duplicate are used as the performance metrics. The configuration of Cluster SD-WSN for homogeneous network, heterogeneous network, and dynamic network are shown in Table 3-4, Table 3-5, and Table 3-6. Then, the result from Cluster SD-WSN is compared with the result from WSN (one sink, 14 sensor nodes) and SD-WSN (one *SDN controller nodes*, 14 *SDN sensor nodes*).

4-6-1 Packet Delivery Ratio of Cluster SD-WSN

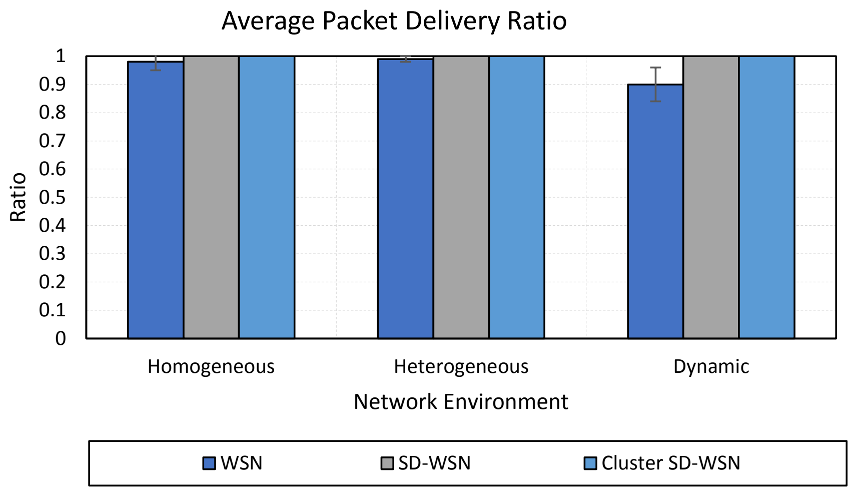


Figure 4-11: Average Packet Delivery Ratio
(WSN, SD-WSN, Cluster SD-WSN)

Figure 4-11 shows the packet delivery ratio between WSN, SD-WSN, and Cluster SD-WSN. Based on the simulation result of Cluster SD-WSN on the homogeneous network, heterogeneous network, and dynamic network, multiple controllers do not affect the performance of Cluster SD-WSN because the result of average PDR for SD-WSN and Cluster SD-WSN is similar, i.e. 1. It indicates that all the packets have been received successfully by the *SDN controller nodes* in a homogeneous network, heterogeneous network, and dynamic network.

4-6-2 Packet Duplication of Cluster SD-WSN

Figure 4-12 shows that clustering controller mechanism achieves to reduce the multiple receptions of a single packet at the *SDN controller node*. In particular, In a homogeneous network, packet duplication decreased by 5%, from 56% to 51%. In a heterogeneous network, packet duplication condition decreased by 10%, from 26% to 16%. Finally, in a dynamic network, clustering controller successfully reduce the packet duplication by 13%, from 41% to 28%.

Clustering controllers can reduce packet duplication because it minimizes the number of multiple hops that must be passed by each *SDN sensor node*. The more hops traversed, the more likely multiple receptions of a single packet at the *SDN controller node*. With multiple controllers, each *SDN sensor node* is more adaptable in dynamic network. Thus, packet duplication can be reduced to 13%.

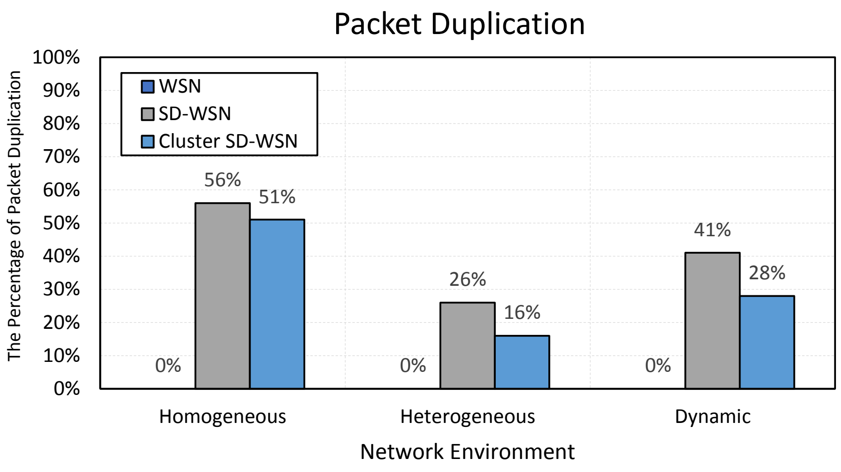


Figure 4-12: Packet Duplication
(WSN, SD-WSN, Cluster SD-WSN)

4-6-3 Average Delay of Cluster SD-WSN

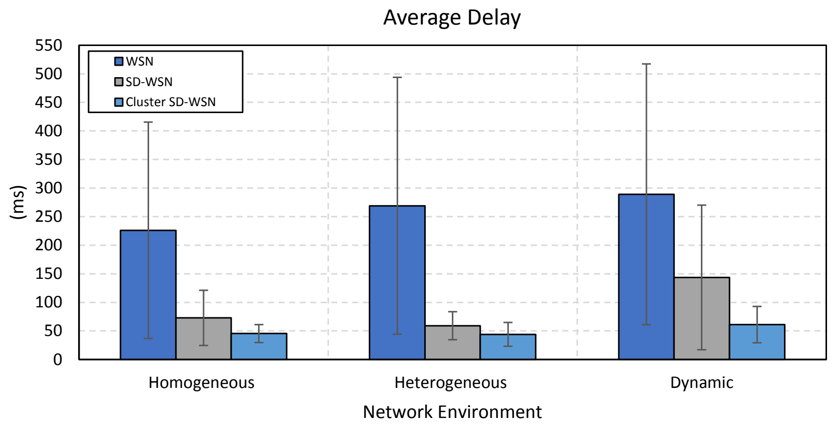


Figure 4-13: Average Delay
(WSN, SD-WSN, Cluster SD-WSN)

Figure 4-13 shows that clustering controller mechanism achieves to reduce the average delay of SD-WSN. In homogeneous network, using Cluster SD-WSN, the average delay and standard deviation of SD-WSN are reduced by 38% and 67%, respectively. In heterogeneous networks, the average delay and standard deviation of SD-WSN are reduced by 26% and 15%, respectively. Furthermore, in dynamic network, clustering controller succeeded in decreasing the percentage of average delay and standard deviation of SD-WSN significantly, that is 57% and 75%.

Since in cluster SD-WSN, the *SDN sensor nodes* are using multiple controllers, instead of single *SDN controller nodes*. Thus, the duration of transmit packet from *SDN sensor nodes* to *SDN control nodes* becomes shorter as the path for each *SDN sensor node* is defined by the nearest *SDN controller nodes*.

4-6-4 Duty Cycle of Cluster SD-WSN

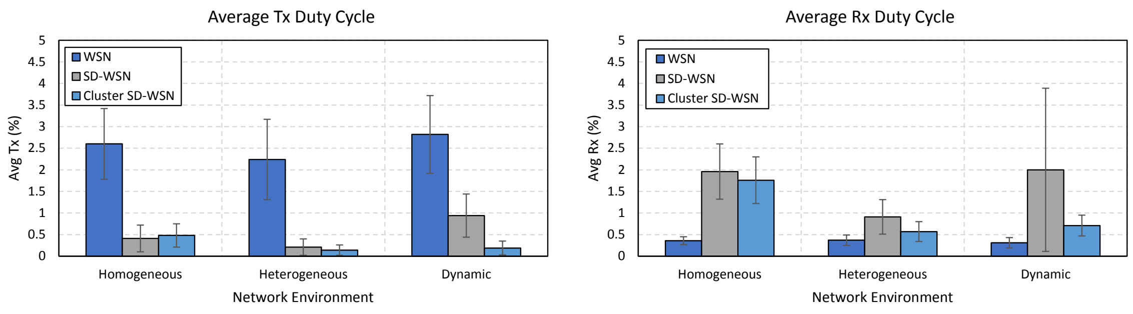


Figure 4-14: Average Duty Cycle
(WSN, SD-WSN, Cluster SD-WSN)

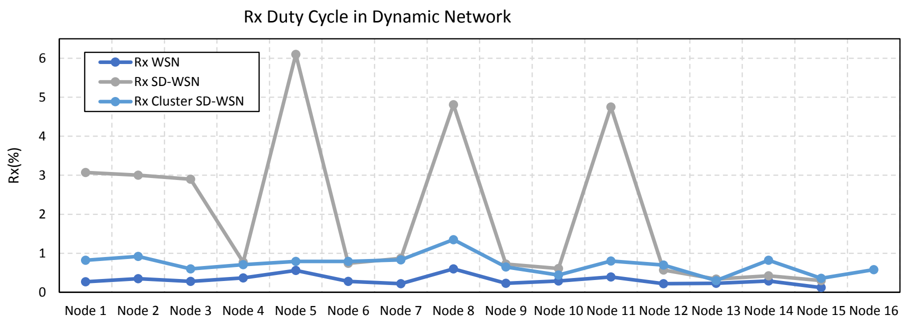


Figure 4-15: Average Duty Cycle per Sensor Node
in Dynamic Network

Duty Cycle on SD-WSN is closely related to the number of *SDN sensor nodes*, network conditions, and packet duplication. The fewer number of *SDN sensor nodes* and the less packet duplication, the smaller the duty cycle of SD-WSN. On average, clustering controller reduces the average duty cycle of SD-WSN, shown in Figure 4-14. In a dynamic network, clustering controller reduces the average duty cycle of Tx and Rx significantly. The average of Tx duty cycle drops about 80%, while the average of Rx duty cycle drops by 64%. Figure 4-15 shows the average of Rx duty cycle for each node in WSN, SD-WSN, and Cluster SD-WSN under dynamic network. Before using clustering controller, fluctuation occurs when nodes 5, 8, and 11 are leaving and rejoining network, but after using clustering controller, the Rx duty cycle value from SD-WSN becomes relatively stable.

Chapter 5

Conclusion and Future Work

This final chapter gives a summary of the most important conclusions drawn from the discussion in this master thesis. After that, the contribution of this thesis and the recommendations that will be useful for future work are described.

5-1 Conclusion

In this thesis, the SDN framework for WSN is evaluated, our implementation of this mechanism shows that Packet Delivery Ratio (PDR) for WSN and SD-WSN are relatively similar around 0.98 to 1 for a homogeneous and heterogeneous network. However, in a dynamic network, the PDR of WSN decreases from 0.98 to 0.9, whereas the average PDR of SD-WSN stays 1. The reason why the PDR of SD-WSN is always 1 is due to the packet duplication. On the one hand, packets duplication can increase the packet delivery ratio because its redundancy can prevent packet drops, on the other hand, packet duplication will increase the load of the network. The packet duplication occurs significantly in the dynamic network, when the number of *SDN sensor nodes* is increasing from 10 to 15, the duplicate packet rises by 33%.

Compared to WSN, SD-WSN performs better in the sense of average duty cycle (Tx) and average delay, since the *SDN controller nodes* reduces the active time of a *SDN sensor node* to find the best route. However, utilizing *SDN controller nodes* increases the average duty cycle for Rx because of packet duplication and dependency with *SDN controller nodes*. Using the centralized concept on the *SDN sensor nodes* causes the *SDN sensor nodes* to become highly dependent on the controller. Thus, in a dynamic network, the *SDN sensor nodes* require a longer adaptation. The indication that SD-WSN is not suitable for a dynamic network is shown by the increasing ratio of average delay for an SD-WSN which rises from 2.03 to 2.3. On the other hand, the ratio of average delay for WSN is only around 1.6 to 1.98. Therefore, SD-WSN is not optimally applied to dynamic conditions as the activity changes from the *SDN sensor nodes* will affect the performance of SD-WSN significantly.

The performance of SD-WSN in a heterogeneous, homogeneous, and dynamic network is relatively worse than WSN, in the sense of packet duplication and Rx duty cycle. Also, SD-WSN is not optimally applied to dynamic conditions as the activity changes from the *SDN sensor nodes* will affect the performance of SD-WSN significantly. To reduce the load on the centralized controller, clustering controller is used to distribute the load on multiple controllers, based on the evaluation, after using clustering controller, SD-WSN performance has increased. Packet duplication and average delay in a dynamic network can be reduced by 13% and 57%, respectively. Clustering controller also successfully makes Rx duty cycle of SD-WSN relatively stable compared to before using clustering controller.

5-2 Contributions

The contributions of this thesis can be divided into the following:

- The performance of SD-WSN and WSN were evaluated in three network environments: homogeneous, heterogeneous, and dynamic networks.
- Packet delivery ratio, average delay, duty cycle, and packet duplication are proposed as the performance metrics for WSN and SD-WSN.
- A clustering controllers are proposed to improve the performance of SD-WSN.
- Data collection protocols (collection tree protocol and rime data collection) were used to measure the performance of SD-WSN and WSN in terms of collecting information from each mote to the *SDN controller node*/sink

5-3 Future Work

- Performance evaluation of SDN framework for WSN is using Cooja as a network simulator. We expect that in future work, the performance evaluation can be tested in realistic scenarios, so the results from our simulation can be compared with the results from the real implementation. A realistic scenario will give a better understanding of the challenges and benefits. Moreover, the framework that is used in this thesis can be implemented into WSN devices directly since the TelosB firmware used in Cooja is the emulator of the actual TelosB firmware.
- CTP is used as the routing protocol for SD-WSN and compare it with Rime data collection for WSN; Integrating the SD-WSN framework with a different routing protocol and evaluating the performance with a different scenario, e.g. mobility might give a better understanding about the effect of SDN on WSN.
- Clustering controller reduces the load of a centralized controller. However, the clustering controllers have not been able to communicate with each other. The development of communication protocol for supporting *SDN controller nodes* is expected to be able to increase scalability and management of a large network.

Bibliography

[1] A. C. G. Anadiotis et al. "Towards a software-defined Network Operating System for the IoT". In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. Dec. 2015, pp. 579–584. DOI: 10.1109/WF-IoT.2015.7389118.

[2] Divith Aruni Babu. "SDN-based WSN Routing Protocol". PhD thesis. State University of New York at Stony Brook, 2016.

[3] Pankaj Berde et al. "ONOS: Towards an Open, Distributed SDN OS". In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. HotSDN '14. Chicago, Illinois, USA: ACM, 2014, pp. 1–6. ISBN: 978-1-4503-2989-7. DOI: 10.1145/2620728.2620744. URL: http://doi.acm.org/10.1145/2620728.2620744.

[4] N. Bizanis and F. A. Kuipers. "SDN and Virtualization Solutions for the Internet of Things: A Survey". In: *IEEE Access* 4 (2016), pp. 5591–5606. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2607786.

[5] *CC2420 (NRND)*. URL: http://www.ti.com/product/CC2420.

[6] Younghwan Choi, Yunchul Choi, and Yong-Geun Hong. "Study on coupling of software-defined networking and wireless sensor networks". In: *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*. July 2016, pp. 900–902. DOI: 10.1109/ICUFN.2016.7536926.

[7] International Electrotechnical Commission. *White Paper Internet of Things: Wireless Sensor Networks*. 2014. URL: http://www.iec.ch/whitepaper/internetofthings/.

[8] contiki-os. *contiki-os/contiki*. URL: https://github.com/contiki-os/contiki/wiki/Radio-duty-cycling.

[9] S. Costanzo et al. "Software Defined Wireless Networks: Unbridling SDNs". In: *2012 European Workshop on Software Defined Networking*. Oct. 2012, pp. 1–6. DOI: 10.1109/EWSDN.2012.12.

[10] Adam Dunkels, Fredrik Österlind, and Zhitao He. "An adaptive communication architecture for wireless sensor networks". In: *Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM. 2007, pp. 335–349.

[11] Rodrigo Fonseca et al. “Four-Bit Wireless Link Estimation.” In: 2007.

[12] H. Fotouhi et al. “SDN-TAP: An SDN-based traffic aware protocol for wireless sensor networks”. In: *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*. Sept. 2016, pp. 1–6. DOI: 10.1109/HealthCom.2016.7749527.

[13] A. Al-Fuqaha et al. “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”. In: *IEEE Communications Surveys Tutorials* 17.4 (June 2015), pp. 2347–2376. ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2444095.

[14] L. Galluccio et al. “SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks”. In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. Apr. 2015, pp. 513–521. DOI: 10.1109/INFOCOM.2015.7218418.

[15] A. De Gante, M. Aslan, and A. Matrawy. “Smart wireless sensor network management based on software-defined networking”. In: *2014 27th Biennial Symposium on Communications (QBSC)*. June 2014, pp. 71–75. DOI: 10.1109/QBSC.2014.6841187.

[16] David Gay et al. “The nesC language: A holistic approach to networked embedded systems”. In: *Acm Sigplan Notices*. Vol. 38. 5. ACM. 2003, pp. 1–11.

[17] Omprakash Gnawali et al. “Collection Tree Protocol”. In: *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys’09)*. Berkeley, CA, USA, Nov. 2009.

[18] I. T. Haque and N. Abu-Ghazaleh. “Wireless Software Defined Networking: A Survey and Taxonomy”. In: *IEEE Communications Surveys Tutorials* 18.4 (May 2016), pp. 2713–2737. ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2571118.

[19] C. Jacquemod, B. Nicolle, and G. Jacquemod. “WSN for smart building application”. In: *10th European Workshop on Microelectronics Education (EWME)*. May 2014, pp. 102–105. DOI: 10.1109/EWME.2014.6877405.

[20] Sushant Jain et al. “B4: Experience with a globally-deployed software defined WAN”. In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 3–14.

[21] D. Kreutz et al. “Software-Defined Networking: A Comprehensive Survey”. In: *Proceedings of the IEEE* 103.1 (Jan. 2015), pp. 14–76. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2371999.

[22] Hedi Krishna. “Providing End-to-end Bandwidth Guarantees with OpenFlow”. In: (2016).

[23] P. Levis et al. “TinyOS: An Operating System for Sensor Networks”. In: *Ambient Intelligence*. Ed. by Werner Weber, Jan M. Rabaey, and Emile Aarts. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 115–148. ISBN: 978-3-540-27139-0. DOI: 10.1007/3-540-27139-2_7. URL: http://dx.doi.org/10.1007/3-540-27139-2_7.

[24] S. Li. *Network Lifetime Analysis of Data Collection Protocols*. Aug. 2014. URL: https://repository.tudelft.nl/islandora/object/uuid:bc28e3ec-4c06-4224-9b19-7afbfd7c93f3?collection=education.

[25] T. Luo, H. P. Tan, and T. Q. S. Quek. “Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks”. In: *IEEE Communications Letters* 16.11 (Nov. 2012), pp. 1896–1899. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2012.092812.121712.

[26] Nick McKeown et al. “OpenFlow: Enabling Innovation in Campus Networks”. In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69–74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. URL: http://doi.acm.org/10.1145/1355734.1355746.

[27] R. Nath. “A TOSSIM based implementation and analysis of collection tree protocol in wireless sensor networks”. In: *2013 International Conference on Communication and Signal Processing*. Apr. 2013, pp. 484–488. DOI: 10.1109/iccsp.2013.6577101.

[28] Musa Ndiaye, Gerhard P Hancke, and Adnan M Abu-Mahfouz. “Software Defined Networking for Improved Wireless Sensor Network Management: A Survey”. In: *Sensors* 17.5 (2017), p. 1031.

[29] Amy Nordrum. *Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated*. Aug. 2016. URL: https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated.

[30] B. T. de Oliveira and C. B. Margi. “Distributed control plane architecture for software-defined Wireless Sensor Networks”. In: *2016 IEEE International Symposium on Consumer Electronics (ISCE)*. Sept. 2016, pp. 85–86. DOI: 10.1109/ISCE.2016.7797384.

[31] B. T. de Oliveira, C. B. Margi, and L. B. Gabriel. “TinySDN: Enabling multiple controllers for software-defined wireless sensor networks”. In: *2014 IEEE Latin-America Conference on Communications (LATINCOM)*. Nov. 2014, pp. 1–6. DOI: 10.1109/LATINCOM.2014.7041885.

[32] B. Trevizan de Oliveira, R. C. A. Alves, and C. Borges Margi. “Software-defined Wireless Sensor Networks and Internet of Things standardization synergism”. In: *2015 IEEE Conference on Standards for Communications and Networking (CSCN)*. Oct. 2015, pp. 60–65. DOI: 10.1109/CSCN.2015.7390421.

[33] Donna O’Shea, Victor Cionca, and Dirk Pesch. “The Presidium of Wireless Sensor Networks - A Software Defined Wireless Sensor Network Architecture”. In: *Mobile Networks and Management: 7th International Conference, MONAMI 2015, Santander, Spain, September 16-18, 2015, Revised Selected Papers*. Ed. by Ramón Agüero et al. Cham: Springer International Publishing, 2015, pp. 281–292. ISBN: 978-3-319-26925-2. DOI: 10.1007/978-3-319-26925-2_21. URL: http://dx.doi.org/10.1007/978-3-319-26925-2_21.

[34] Fredrik Osterlind et al. “Cross-level sensor network simulation with cooja”. In: *Local computer networks, proceedings 2006 31st IEEE conference on*. IEEE. 2006, pp. 641–648.

[35] COY. P and GROSS N. *21 Ideas for the 21st Century*. 1999. URL: http://www.businessweek.com/1999/99_35/2121_content.htm.

[36] Georgios Z Papadopoulos et al. “Toward a packet duplication control for opportunistic routing in WSNs”. In: *Global Communications Conference (GLOBECOM), 2014 IEEE*. IEEE. 2014, pp. 94–99.

[37] R. R. Rout and S. K. Ghosh. “Enhancement of Lifetime using Duty Cycle and Network Coding in Wireless Sensor Networks”. In: *IEEE Transactions on Wireless Communications* 12.2 (Feb. 2013), pp. 656–667. ISSN: 1536-1276. DOI: 10.1109/TWC.2012.111412.112124.

[38] B. Rubio, M. Diaz, and J. M. Troya. "Programming Approaches and Challenges for Wireless Sensor Networks". In: *2007 Second International Conference on Systems and Networks Communications (ICSNC 2007)*. Aug. 2007, pp. 36–36. DOI: 10.1109/ICSNC.2007.63.

[39] H Silva et al. "WARM: WSN application development and resource management". In: *XXXIV Simposio Brasileiro de Telecomunicacoes e Processamento de Sinais (SBrT 2016), Brazil*. Aug. 2016.

[40] *Software Defined Networking (SDN) - The seven benefits of software-defined networking*. Apr. 2017. URL: https://www.cisco.com/c/en/us/solutions/software-defined-networking/benefits.html.

[41] *Software-Defined Networking (SDN) Definition - Open Networking Foundation*. URL: https://www.opennetworking.org/sdn-definition/.

[42] *TelosB*. URL: http://tinyos.stanford.edu/tinyos-wiki/index.php/TelosB.

[43] F. Wang and J. Liu. "Duty-Cycle-Aware Broadcast in Wireless Sensor Networks". In: *IEEE INFOCOM 2009*. Apr. 2009, pp. 468–476. DOI: 10.1109/INFCOM.2009.5061952.

[44] Jiliang Wang et al. "On the delay performance analysis in a large-scale wireless sensor network". In: *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*. IEEE. 2012, pp. 305–314.

[45] Yanwen Wang et al. "An energy-efficient SDN based sleep scheduling algorithm for WSNs". In: *Journal of Network and Computer Applications* 59 (2016), pp. 39–45. ISSN: 1084-8045. DOI: http://dx.doi.org/10.1016/j.jnca.2015.05.002. URL: http://www.sciencedirect.com/science/article/pii/S1084804515000910.

[46] Yun Wang et al. "Intrusion detection in homogeneous and heterogeneous wireless sensor networks". In: *IEEE transactions on mobile computing* 7.6 (2008), pp. 698–711.

[47] W. Xiang, N. Wang, and Y. Zhou. "An Energy-Efficient Routing Algorithm for Software-Defined Wireless Sensor Networks". In: *IEEE Sensors Journal* 16.20 (Oct. 2016), pp. 7393–7400. ISSN: 1530-437X. DOI: 10.1109/JSEN.2016.2585019.

[48] D. Zeng et al. "Energy Minimization in Multi-Task Software-Defined Sensor Networks". In: *IEEE Transactions on Computers* 64.11 (Nov. 2015), pp. 3128–3139. ISSN: 0018-9340. DOI: 10.1109/TC.2015.2389802.

[49] J. Zhou et al. "SDN-Based Application Framework for Wireless Sensor and Actor Networks". In: *IEEE Access* 4 (2016), pp. 1583–1594. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2547890.