# MANAGEMENT OF THE INTERNET OF THINGS (IOT) WITH SOFTWARE-DEFINED NETWORKING

CEM TÜRKER
B.S., Computer Engineering, IŞIK UNIVERSITY, 2019

Submitted to the Graduate School of Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Engineering

IŞIK UNIVERSITY
2019

IŞIK UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

# MANAGEMENT OF THE INTERNET OF THINGS (IOT) WITH SOFTWARE-DEFINED NETWORKING

CEM TÜRKER

# MANAGEMENT OF THE INTERNET OF THINGS (IOT) WITH SOFTWARE-DEFINED NETWORKING

## Abstract

In recent years, Software Defined Networking is started to shape telecommunication industry. Control plane and data plane separation is applied on telecommunication infrastructure and have evolved packet network. There are also many studies about the application of Software Defined Networking on Wireless Sensor Networks domain. One of them is SDN-WISE that aims to reduce the complexity of network configuration and management of Wireless Sensor Networks. In this thesis, a couple of contributions are done on SDN-WISE architecture to improve the network performance by changing the routing mechanism.

In wireless sensor networks, nodes, who are close to sink, places a critical role, because those nodes are being frequently selected to reach destination path when a data is required to deliver on edge nodes in the topology. In this thesis, we try to maximize the lifetime of the network by selecting destination paths considering forecasted remaining battery level. Lifetime defined as follows. When any node dies in a topology, it is assumed that the lifetime of the network is over. This approach is implemented on SDN-WISE architecture and to see the performance of this method a set of experiments are performed.

This thesis is about maximizing the lifetime of the network on SDN-WISE by applying forecasting on node's battery level. Architectural modifications is done and linear regression is added to reach the objective. Our results show that the lifetime of the network is increased. However, while increasing the lifetime of the network, the packet delivery delay is increased between the nodes. We analyse this trade-off.

**Keywords: SDN-WISE, WSN, Routing, Forecast, Linear Regression**

# YAZILIM TABANLI AĞLAR İLE NESNELERİN İNTERNETİ (IOT) YÖNETİMİ

## Özet

Son yıllarda, Yazılım Tanımlı Ağ, telekomünikasyon endüstrisini şekillendirmeye başlamıştır. Kontrol yollari ve veri yollari ayrımı telekomünikasyon altyapısına uygulanır ve yazilim ağlarini geliştirmiştir. Kablosuz Algılayıcı Ağlarda Etki Alanında Yazılım Tanımlı Ağların uygulanması hakkında birçok çalışma vardır. Bunlardan biri, ağ konfigürasyonunun ve Kablosuz Sensör Ağlarının yönetiminin karmaşıklığını azaltmayı amaçlayan SDN-WISE platformudur. Bu tezde, ağ performansını artırmak amacı ile yönlendirme mekanizmasını değiştirmek için SDN-WISE mimarisine birkaç katkı yapılmıştır.

Kablosuz algılayıcı ağlarda, alıcıya yakın olan düğümler kritik bir rol oynar, çünkü bu düğümler, topolojideki kenar düğümlerde bir veri istendiğinde, hedef yola ulaşmak için sık sık seçilmektedir. Bu tezde, kalan pil seviyesini göz önünde bulundurarak hedef yolları seçerek ağın ömrünü maksimize etmeye çalışıyoruz. Ömür boyu; herhangi bir düğüm bir topolojide öldüğünde, ağın kullanım ömrünün sona erdiği varsayılır. Bu yaklaşımı SDN-WISE mimarisine uyguladik ve bu yöntemin performansını görmek için bir dizi deney yaptik.

Bu tez, düğümün pil seviyesi ile ilgili tahminleri uygulayarak ağın kullanım ömrünü SDN-WISE'de maksimize etmekle ilgilidir. Bunun için SDN-WISE'de mimari değişiklikler yapılmış ve hedefe ulaşmak için lineer regresyon eklenmiştir. Sonuçlar ağın kullanım ömrünün arttığını göstermektedir. Bununla birlikte, ağın ömrünü arttırırken, düğümler arasında paket iletim gecikmesi artmıştır. Bu ödünleşim analizi sonuçlarda paylaşılmıştır.

**Anahtar kelimeler: SDN-WISE, WSN, Yönlendirme, Önden Tahmin, Lineer Regresyon**

# Acknowledgements

First of all, I want to thank to Assist. Prof. Dr. Ayşegül Erman Tüysüz, who has supervised me during my thesis and has helped me to improve my work.

Also, I have special thanks to Cansu Toprak, who has supported me while working on my thesis.

Finally, I thank my lovely parents and brother, Mehmet Türker, Işıl Türker and Can Türker, who supported me all the time.

*To My Family…*

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| **IoT** | **I**nternet **o**f **T**hings |
| **WSN** | **W**ireless **S**ensor **N**etwork |
| **SDN** | **S**oftware **D**efined **N**etworks |
| **SD-WSN** | **S**oftware **D**efined **W**ireless **S**ensor **N**etworks |
| **PF** | **P**article **F**iltering |
| **PID** | **P**rocess **ID** |
| **VM** | **V**irtual **M**achine |
| **RPL** | IPv6 **R**outing **P**rotocol for **L**ow Power And Lossy Networks |
| **RSSI** | **R**eceived **S**ignal **S**trength **I**ndicator |
| **RPC** | **R**emote **P**rocedure **C**all |

# Chapter 1

# Introduction

In this thesis, Software Defined Networks (SDN) has investigated on Wireless Sensor Networks (WSN) to obtain maximum network lifetime using the routing in an efficient way. Lifetime means that, if any node's battery dies in the topology, it is assumed that lifetime of the network is over. SDN plays an important role to achieve this goal and thanks to it's centralized and programmable control unit. Resource and process control is difficult to manage for WSN due to behaviours and structures of the sensors. SDN aims to provide applicable controlling mechanism for WSN. Most of the studies have already changed their scope into SD-WSN topic [1].

At 2020, 50 billion devices are measured to be connected to the network and most of them will be actuators and the sensors [2]. Thus, monitoring them and intervene the routing from centralized management is needed to gain maximum resource usage. That is why WSN integration with SDN is important.

One of the SD-WSN approach, which is SDN-WISE, has investigated and used in this thesis. A particular solution has proposed as an improvement on SDN-WISE protocol. The solution is about prediction of the next battery usages for the future using linear regression and applying the forecasted battery levels on a rerouting algorithm. Battery level plays a significant role in WSN, since the nodes are mostly driven by nonchargeable batteries [3].

In the thesis, it is shown that SD-WSN network resources can be managed and battery level of the nodes in the topology can be balanced using rerouting by the controller unit. Thus, the weight over the node will be reduced, and the battery consumption will be degraded as well.

Tests are driven towards different size of WSN topologies. Lifetime of a topology and communication delay are measured and comparison have made with the proposed and the default SDN-WISE protocol. As a result, longer lifetime has achieved for every topology; however, communication delay was increased.

# Chapter 2

# Background

## 2.1 Previous Works

In recent years, telecom industry has started to shift their infrastructures to SDN for their future plans. The reason is to have a centralized view and supporting programmability of protocols and network functions by decoupling data plane and controller plane. Currently data plane and control plane are mostly integrated in network devices [4].

There are many different Software Defined Wireless sensor solutions have proposed already. In this study, we have worked with SDN-WISE application. The reason is controller unit has already implemented in the application. Furthermore, node and sink codes are open-sourced which means that they can be extended by additions. SDN-WISE also provides testing on network using Cooja. The tutorial has been shared for testing through their website [5]. Cooja simuation tool will be described in the Section 3.2.1 briefly.

SDN-Wise is a proposed application for SD-WSN and has its own protocol between control and network layer like Openflow. This is for communicating with sensors and called as SDN-WISE Protocol. The SDN-WISE controller architecture is nearly similar to how SDN controller is working. Only differences are topology discovery, network packet processing and topology management phases [6]. The details of them will be discussed in Chapter 2.

Studies have been applied to SDN-WISE to test functionality, adding big data integration and increase lifetime of the network. These are adding the trickle timing algorithm and adding a LSTM-ANN to SDN-WISE.

SDN-WISE is proposed for managing easily the WSN with functionality of SDN. It means that, it provides a observably and management overall the network. Also their proposed solution allows different WSN networks to work together. To do that, they have proposed a different stack implementation for WSN nodes [7].

Big data integration with SDN-wISE is also proposed. The map-Reduce functionality is added to SDN-WISE, and different data-aggregation is applied in their tests [8]. They compared performance of their proposed system with three different topologies. Also, their data process is done at different levels; one is at cloud based, in WSN node, other-one is hybrid.

Trickle timing algorithm is integrated with SDN-WISE to improve energy efficiency. The trickle algorithm allows nodes in low-power and lossy networks (the IPv6 Routing Protocol) to share information in a highly robust, scalable manner and simple way [9]. When the network is consistent and stable, the packet frequency will be low; however, when a network inconsistency is detected, the number of packet rate will be increased between nodes to make it consistent again. According to their results, they gain better efficiency over battery consumption on nodes. They are referring that their approach performed higher compared to RPL in terms of radio duty cycle. The tests are applied with 5 nodes and 1 sink which is really insufficient for testing the performance.

LSTM-ANN is added to controller plane to predict the future flows [10]. This is the first study that machine learning have applied. The algorithm tries to find a pattern from existing network and passes it to a routing algorithm and tries to predict a future move. In their test results, they discussed the packet delivery by comparing existing shortest path algorithm (Dijkstra) and their modification. In addition to that, they share how much their packet predictions are matched with expected flows.

## 2.2 Software Defined Networking (SDN)

This section describes an overview of Software Defined Networking, which is developing rapidly at present. With new developments such as on IoT, network virtualization and big data, it seems that SDN will play a significant role on management of network resources. Traditional networks have limitations in order to meet the requirements of this growing area. Therefore, most of the researches that are in the same field have been turned their perspectives towards SDN. IoT can also be accepted as a subsection for WSN. Therefore, understanding of SDN in WSN field will help to focus on system requirements from the basic. SDN architecture is shown in Figure 2.1 [11].

### 2.2.1 SDN Architecture

Unlike traditional networks, SDN decouples controlling and forwarding planes. It introduces centralized management for network elements in the topology. This centralized management is called as controller for SDN architecture and it is purely programmable. By centralizing network in control plane, it provides infrastructure to SDN applications which can be security, monitoring, optimizing or configuring specific. It also makes networking devices vendor specific [12].

Infrastructure layer is responsible for forwarding data packets based on instructions. These instructions are received from control layer. Some protocols, Openflow is commonly known, help to communicate between these layers. In this thesis, we used special protocol for that, SDN-WISE protocol. In the control layer, decisions are made on packet forwarding. It basically behaves forwarding table for devices. It maintains packet's actions and based on algorithms, it can also decides routing path. SDN applications are located in application layer. These applications basically can operate behavior of forwarding on demand.

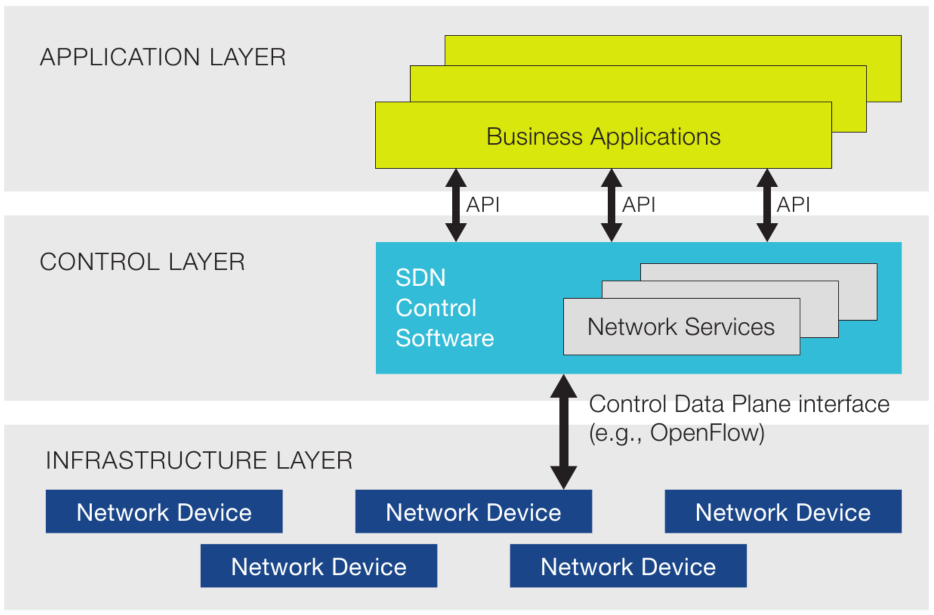

Figure 2.1: SDN Architecture Overview.

### 2.2.2 SDN Features

SDN has the following features:

- More granular architecture for network area
- Programmable control layer which helps to monitor, scale and manage
- Vendor independent network elements
- Increased network reliability and security
- Rapid delivery of new network features

### 2.2.3 SDN Capabilities from WSN Point of View

Distributed resource management, monitoring and adding an intelligence to network adds a value to WSN. In addition to that, there are many different type of WSN topology types like query driven or mobile and they require different type of routing algorithm. SDN provides these required feature to WSN. From SDN

Controller, it is possible to trace network resource and do required actions. Moreover, in controller, it is possible to switch or deploy different kind of algorithm and apply them on WSN topology. Even it is easy to integrate machine learning approach. It requires only one application deployment to do. From WSN point of view, SDN provides a really good features to provide long support. However, these changes require trade-off. For instance, to observe resources, there should be some report packet going through the nodes and it will cause them to loose battery. SDN-WISE provides the integration between SD-WSN.

## 2.3 SDN-WISE

In SDN, management of the network is done from centralized place and control plane and data plane is physically separated. SDN Controllers send the flow rules to the Openflow switches to apply packet classification and forwarding packets throughout the network topology. When switch does not have the any information about incoming packet, it asks assistance from controller and controller applies the flow rules to the switches. Controller gives the decision about its policies. These policies could be modified with installing new controller software. The SDN-WISE uses this approach in wireless infrastureless networks[6]. The flow rules are named as Wise Flow rule. Openflow protocol is named as SDN-WISE protocol. Assigned Wise flow rules are tracked in Wise Flow table in nodes. In SDN-Wise shortest path between nodes is calculated by using Dijkstra algorithm as a default. The SDN-WISE architecture is shown in Figure 2.2 [13].

### 2.3.1 Protocol Architecture

SDN-WISE networks consists of Sensor-nodes and one or multiple sink. Sink is the gateway to the controller. It plays as a bridge between data plane and controller plane. Nodes who needs to send data to controller needs to go through the controller or vice versa. The communication between sink and controller

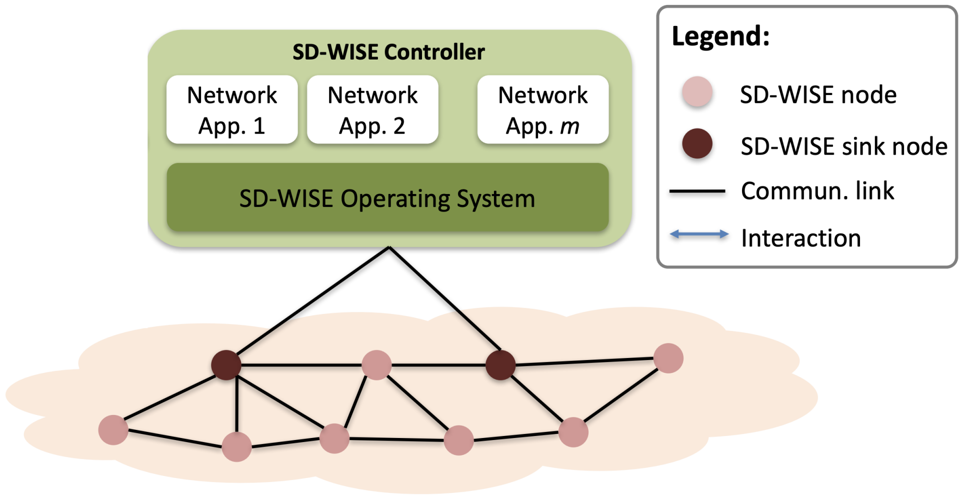

Figure 2.2: SDN-WISE Architecture.

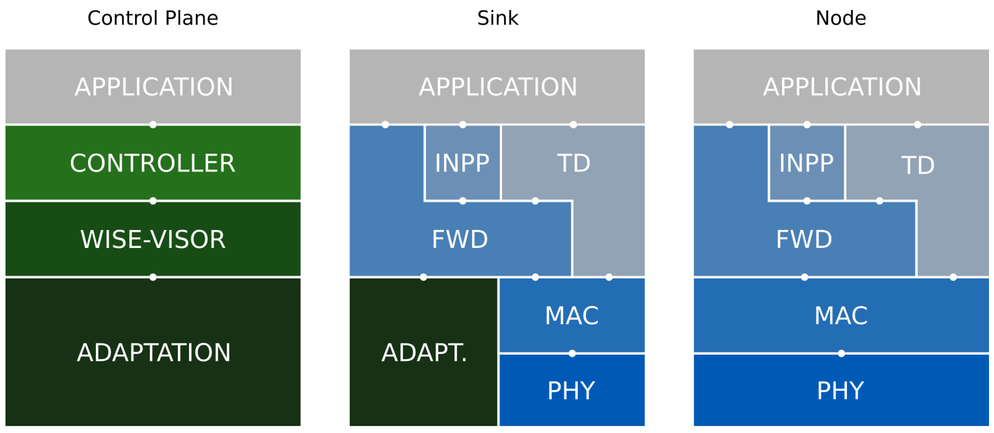

Figure 2.3: SDN-WISE Controller, Sink And Node Network Layers.

represents the controller plane, between sink and nodes represents the data plane. The protocol architecture of SDN-WISE is shown in Figure 2.3 [14].

SDN-WISE uses IEEE 802.15.4 on physical and MAC layers. Network elements are sink and node. The difference between sink and node is that sink is connected with network interface to infrastructured network. Thus that all control packets should go through the sink to leave the WSN or reach the controller [14].

Incoming packets are handled in forwarding (FWD) layer as specified in the Wise Flow table. The wise flow tables are updated according to Controller plane directives.

In network packet processing (INPP) layers is responsible for data aggregation or other in network processing. Also, this do some optimization, it chains similar packets that must be sent to similar destination according to packet size.

If the incoming packet is not matching with Wise Flow table, the request is sent to controller. While packets is sent to controller, it must go through a sink. In order to reach sink, every node must know the efficient way to reach sink, in other words, best next hop through the sink. The best next hop through the sink is calculated using its neighbor nodes using topology discovery (TD).

Network logics or policies are demanded by one or several controllers and a Wise-visor. The wise-visor layer abstracts the network resources so that different demanded request from several controllers, can run over the same set of physical devices.

Adaptation layer is used for formatting messages which are received from a Sink, so that they can be handled by wise-visor layer or vice-versa

# Chapter 3

# Proposed Routing Architecture and Test Environment

In this thesis, the aim is to maximize the lifetime of a WSN network. Lifetime of a network can be defined differently. One of the most commonly used definitions of network lifetime is the time at which the first sensor node goes out of battery and cannot send/receive packets. Another definition is the operational time of the network during which it is able to access all nodes and not disconnected. The most optimistic definition can be the time until all nodes die.

In this thesis, the lifetime of a network is defined such that if any node goes out of battery, lifetime of the network is over. Thus, we choose "time until the first node dies" definition. Battery plays a significant role in terms of maximizing the lifetime; however, current implementation of SDN-WISE has not considered battery in shortest path algorithm (Dijkstra algorithm). They preferred to use only received signal strength indicator (RSSI) value in algorithm. As a solution in this work, the battery level is considered and applied on the routing algorithm.

In the new routing algorithm, battery levels are forecasted and applied. In order to do that, a new module is written and integrated with existing SDN-WISE controller.

At last, testing environment has already been defined in SDN-WISE using Cooja, but since existing code base has changed, libraries, which are used in Cooja, are needed some modifications due to lack of incompatible library versions. There is

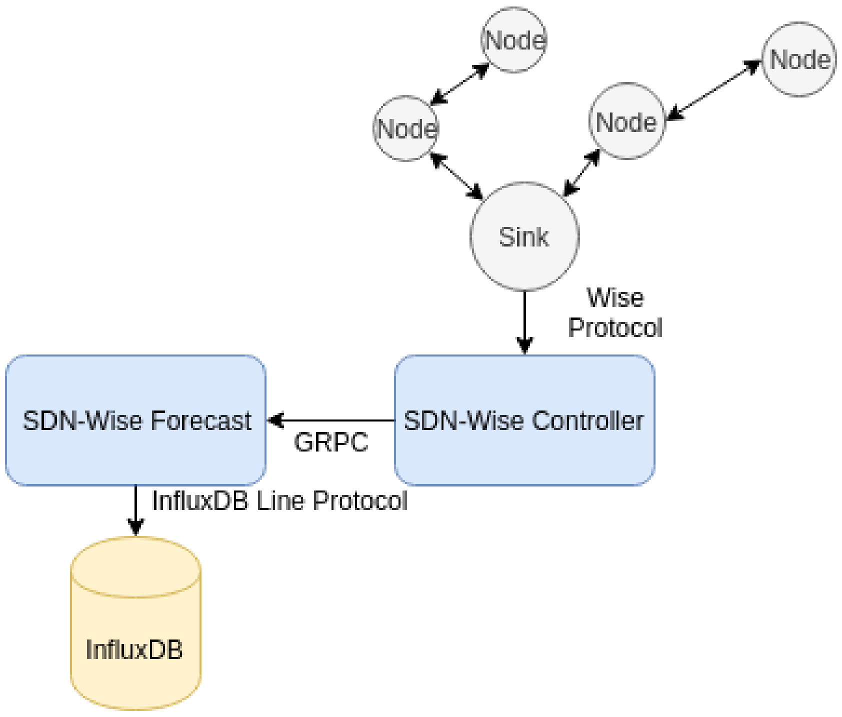

Figure 3.1: Forecasting Architecture Overview.

no defined way to do automated tests in SDN-WISE using Cooja VM. Thus, it is an other contribution defined in this thesis work.

## 3.1 Architecture

### 3.1.1 Overview

The proposed architecture consists of three main applications which are InfluxDB, SDN-WISE Forecast and SDN-WISE Controller. InfluxDB stores time based battery data and SDN-WISE Controller is the WSN controller unit. SDN-WISE Forecast is a new application which is implemented for forecasting the batteries. It uses InfluxDB as a storage unit and SDN-WISE Controller as a controller. While it is doing rerouting, it queries the forecasted batteries from SDN-WISE Forecast and applies batteries to the routing algorithm. The proposed architecture is shown in Figure 3.1

#### 3.1.1.1 SDN-WISE Controller

SDN-WISE Controller is a standalone application which manages a wireless sensor network with SDN-WISE protocol. It provides centralized management and controls the routing between sensor nodes, battery statuses of the nodes and also it can send actions to nodes through connected sinks. In proposed solution, codes related to battery and routing has been modified. Also, adapter is added to connect SDN-WISE Forecast application using gRPC protocol.

In testing, each report packets which are received from nodes are sent to SDN-WISE Forecast service. Also, at certain periods (40 seconds has chosen to have enough number of report packets), forecasted batteries are gathered from SDN-WISE Forecast service. The forecasted batteries are used while rerouting when they are needed. When a report packet is received, the network topology is being checked. If more efficient routing path is found, Dijkstra algorithm is being triggered. In forecast scenario, every 40 seconds, battery values are being same for each node because they are being forecasted ones.

Since battery is being involving near RSSI in Dijkstra algorithm, battery and RSSI values are needed to be balanced. Also, these values play important role for the sake of our tests. Despite, most of the configuration parameters already existed such as connection timeout for node; however, RSSI and Battery were not in there. In proposed solution, these parameter are added to the configuration. Sum of these parameters indicates total weight for node. Total weight is accepted as 1.0. If RSSI is set to 1.0 as a weight, it makes battery weight as 0. In another word, SDN-WISE Controller is going to behave as default (without forecasting service). However, if RSSI weight is set to 0.5, this also sets battery weight to 0.5 and it triggers forecasting service. Weight indication was required for performance measurement under different weights (like RSSI as 0.5 and battery as 0.5). In addition, it is important to consider that every node starts with same battery size. Maximum value for battery size is mapped to 255 by the SDN-WISE Controller

```
message ForecastResponse {
    repeated ForecastNode nodes = 1; //list of nodes
}

message ForecastNode {
    string nodeID = 1; //Node identifier
    int64 estimatedBattery = 2; // Forecasted battery
    int64 time = 3; // forecasted time
}
```

Figure 3.2: Forecast Response Message.

```
message StatSaveRequest {
    string nodeID = 1; //Node identifier
    int64 currentBattery = 2; //Battery value that is
                             // captured from report package
}
```

Figure 3.3: Report packet delivery Message To SDN-WISE Forecast.

to have a numerical display. It is also same for RSSI. The reason is that these parameters are corresponded to 1 byte allocation in the report packet.

For the communication with SDN-WISE Forecast service, gRPC is used. gRPC is a network protocol that proposed by Google. It uses HTTP 2.0 and sends data in binary format. When we compared to HTTP, results show that it is faster and the transferred data size is very small. Also, connection between SDN-WISE Controller and SDN-WISE Forecast is bidirectional. gRPC can be bidirectional, but HTTP can not.

The received forecasted node's message body is represented in Proto file format in Figure 3.2.

SDN-WISE Controller sends its node's report with Figure 3.3 Proto structure to SDN-WISE Forecast.

```
SHOW TAG VALUES FROM Battery WITH KEY = 'Device'
```

Figure 3.4: Retrieve TAG Values From Battery.

```
SELECT BatteryLevel FROM Battery WHERE Device = 'NODE TAG'
AND time >= 'previous_forecast_time' ORDER BY time ASC
```

Figure 3.5: Retrieve Battery Level.

#### 3.1.1.2 SDN-WISE Forecast

SDN-WISE Forecast is a standalone application which is written in Go. The purpose of the application is providing forecasted batteries using linear regression.

When a battery report is received at SDN-WISE Controller, the report message is forwarded to SDN-WISE forecast only if battery weight is bigger than 0. Received reports are stored in InfluxDB using TAG (Node identifier). Each node's reports are stored in **Battery** table. Node's ID is used as a TAG to distinguish nodes in **Measurement** table. Most likely from each node, it is expected to receive 6 report packets before doing forecast. Every 40 seconds, collected batteries are queried from InfluxDB for each node and batteries are forecasted for $current\_time + 40\ seconds$.

When the scheduled period is reached in time task, existing node identifiers are queried with below command from InfluxDB.

Once TAGs are received with Figure 3.4, for each TAG batteries and their recording time are queried(Figure 3.5).

When the batteries and their report time are collected, $current\_time + 40\ second$ is going to be predicted using linear regression.

#### 3.1.1.3 InfluxDB

InfluxDB [15] is an open source time series database. It is written in Go language. It provides high-availability storage and efficient retrieval of time series data in fields such as Internet of Things sensor data, monitoring and real-time analytics. It is recommended by open source communities to monitor nearly real-time or real-time data. There are many alternatives like InfluxDB such as TimescaleDB and OpenTSDB; however, it was chosen in the first place because it was easy to integrate with Grafana. Grafana is a visualization tool to demonstrate data in graphs [16]. However, after couple of decisions, Grafana was no longer needed. Even without Grafana, InfluxDB has been still continued to be used because significant progress has already done about querying and insertion operations. Also, SDN-WISE Forecast service has adopted the InfluxDB Go client too.

InfluxDB is used with Docker. Docker is a containerization tool that allows to maintain applications as a running service [17]. SDN-WISE Forecast service connects to InfluxDB using it's line protocol from 8086 port.

### 3.1.2 Algorithm Modification

In the current SDN-WISE Controller, battery level is not taken into account in Dijkstra algorithm. To apply the proposed solution remaining battery level of a node plays a crucial role. Therefore, remaining battery level value should be considered by the shortest path algorithm.

The library, which SDN-WISE Controller uses to calculate routing, also supports the other parameters like node parameters besides vertex parameters. Thus, the proposed solution has added by using the features of existing library. The Algorithm 1 represents the existing solution in SDN-WISE Controller. The Algorithm 2 represents new algorithm proposed in this work.

In following algorithms, `dist` array represents the RSSI value between nodes. The value $v.weight$ represents the node's battery level. It is important to note that the RSSI value and battery level value are not fully mapped. RSSI value and battery level value range between 0 and 255. Moreover, the weights of these values play an important role in the calculation. The values are multiplied with their weight values before using in the algorithm. For instance, if RSSI weight is 0,7 and battery weight is 0,3 then it will be applied in the algorithm as $v.weight \times 0,3$ and $dist[v] \times 0,7$. Since we work on a shortest path, we are trying to reach lowest value. Thus, while using the battery levels, we are subtracting the obtained value from 255 which is the max battery level value mapped in the controller side. In other words, $v.weight$ is already been subtracted from 255 before multiplied by 0,3.

**Algorithm 1:** Current SDN-WISE Controller Dijkstra Algorithm.

```
Function Dijkstra(Graph, source):
    forall vertex ∈ Graph do
        $dist[v] \leftarrow \infty$    // initial distance from source to vertex v is set to infinite
        $previous[v] \leftarrow undefined$    // Previous node in optimal path from source
    end
    $dist[source] \leftarrow 0$    // Distance from source to source
    $Q \leftarrow Set < n >$    // all nodes in the graph are not optimized thus are in Q
    while $Q.size! = 0$ do
        $u \leftarrow node$    // in Q with smallest in dist
        $Q.remove(u)$
        forall neighbor $v \in u$ do
            $alt \leftarrow dist[u] + dist\_between(u, v)$    // The Link quality is added in this part as dist
            if $alt < dist[v]$ then
                $dist[v] \leftarrow alt$
                $previous[v] \leftarrow u$
            end
        end
    end
    return previous
```

**Algorithm 2:** New Dijkstra Algorithm Including Weights.

```
Function Dijkstra(Graph, source):
    forall $vertex \in Graph$ do
        $dist[v] \leftarrow \infty$    // initial distance from source to vertex v is set to infinite
        $previous[v] \leftarrow undefined$    // Previous node in optimal path from source
    end
    $dist[source] \leftarrow source.battery$    // Distance from source to source
    $Q \leftarrow Set < n >$    // all nodes in the graph are not optimized thus are in Q
    while $Q.size! = 0$ do
        $u \leftarrow node$    // in Q with smallest in dist
        $Q.remove(u)$
        forall $neighbor\ v \in u$ do
            $alt \leftarrow dist[u] + dist\_between(u, v) + v.battery$    // The Link quality is added in this part as dist and weight is added as battery
            if $alt < dist[v]$ then
                $dist[v] \leftarrow alt$
                $previous[v] \leftarrow u$
            end
        end
    end
    return previous
```

| | X Feature Time Readable | X Feature Time In Unix | Y Battery Level |
|---|---|---|---|
| Entry 1 | 2019-08-03 23:10:42 | 1564866642 | 210 |
| Entry 2 | 2019-08-03 23:10:47 | 1564866647 | 208 |
| Entry 3 | 2019-08-03 23:10:52 | 1564866652 | 207 |
| Entry 4 | 2019-08-03 23:10:57 | 1564866657 | 204 |
| Entry 5 | 2019-08-03 23:11:02 | 1564866662 | 199 |
| Predicted | 2019-08-03 23:11:42 | 1564866702 | **179** |

Table 3.1: Battery forecasting Example With Linear Regression.

### 3.1.3 Forecasting Battery Levels

Linear regression is used for forecasting the batteries. Linear regression learning is provided by Sajari's regression library [18]. To be able to use the library, it is needed to provide at least 2 samples. In the solution `X` feature was time of the battery read and `Y` was the battery. To forecast the future time, in the solution `X` feature was given as $current\_time + 40\ seconds$ and expected the predicted `Y` value (battery) for that time. Most likely for each run, sample sets are consisting of 5 because for each node were sending 5 or 6 report packets in 40s (Every 6 second report packets are sent to sink).

While forecasting, `X` feature is given to linear regression in Unix format. In above sample, predicted battery is **179**. For each node in the above, estimation is done every 40 seconds. The sample is shown in Table 3.1

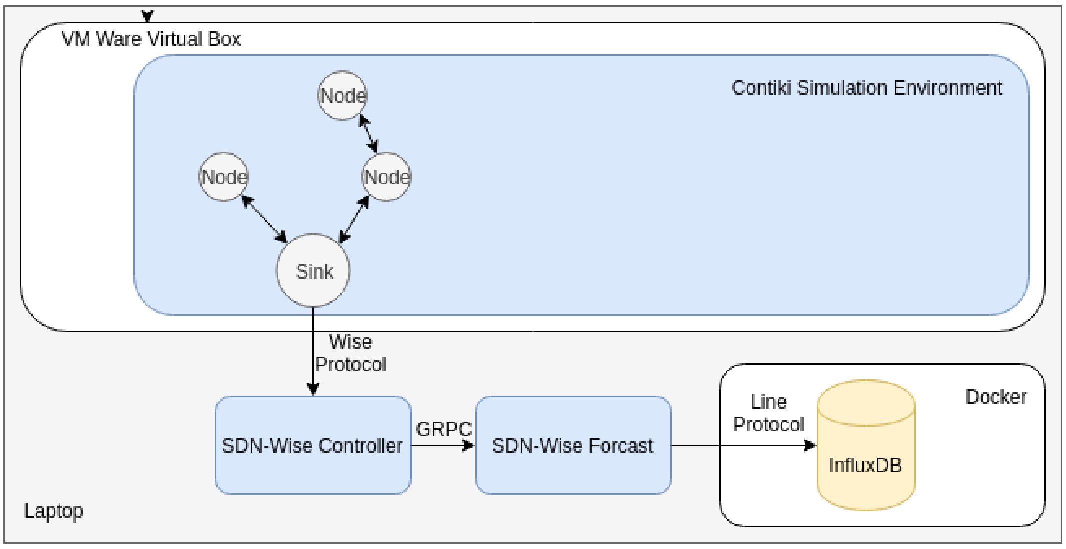

Figure 3.6: Test Architecture Overview.

## 3.2 Test Environment

### 3.2.1 Overview

SDN-WISE test environment is created as recommended from their website. They refers to install VMware to use Cooja simulation environment. The steps that they mentioned in their installation page was followed in this thesis. Addition to these steps, .class files of existing node and sink are updated due to incompatibility with latest SDN-WISE controller version.

The way of the tests are done with following steps; from the topology randomly 3 node is selected and from the controller the queries are sent to randomly selected 3 nodes. Every 1 second, this request is sent to nodes again and again. Until any of the node in topology has reached to 5% of battery. As this happens, the node who reaches 5% battery level sends a report package immediately to the controller. After controller receives package, it stops the application and creates an output file. Output file will be mentioned on the Data Collection and Reporting section.

The test environment architecture is shown in Figure 3.6

### 3.2.2 Test Tools

As testing tools, VMware for running the tests in the machine, Cooja for WSN simulation, Docker for containerization of InfluxDB and personal laptop has been used.

VMware is a virtual machine tool that allows you to run virtual operating system on your computer [19]. The version of VMware that is used in tests is 14.1.1 build-7528167. In VMware snapshot of that already has created image is used and it is provided by SDN-WISE. That image contains 12.04.5 LTS, Precise Pangolin of Linux Ubuntu. While the tests are running, 4GB RAM is shared with VM image from physical partition.

Contiki is an operating system that is used for internet of things devices. It is mostly used for embedded systems and it runs on small microcontroller devices.

Cooja is simulation tool that allows you to do simulate Wireless Sensor Network devices in different scenarios using Contiki. There are many different options available for that, but since SDN-WISE has preferred this simulation tool, it is also preferred by us [20]. Cooja allows you to do WSN simulation from it's graphical user interface or from command line [21]. In the following Test Automation section in the thesis, commands will be mentioned. Cooja version is 2.6 in tests.

Docker is a container that allows you to run your applications in an operation level virtualization [17]. The difference from virtual machines is that it shares kernel with host machine. In the tests, InfluxDB has been ran as docker container. The docker version is 18.09.0 in tests.

Personal laptop is hosted of all those tools. It runs wit Linux operating system. The version is Ubuntu 16.04.5 LTS [22]. It has basically 8GB RAM and Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 8 real core and 12 virtual thread.

### 3.2.3 Test Automation

Each test nearly takes at least 600 second to finish. Thus, it was needed to be automated. To do that, two different scripts are prepared. One scripts was running in VM, other one was running in actual machine.

The script which runs in actual machine, is managing when to run VM script, SDN-WISE Controller and SDN-WISE Forecast. When this script is started, it starts with SDN-WISE Forecast and SDN-WISE Controller first and puts them in a fork and takes their PID id. After that is sends a request to VM script that controller and forecast applications are ready also delivers the topology template that needs to run. When VM scripts is triggered, it waits for taken PID id to terminate itself. The termination is happened when SDN-WISE Controller detects that one of the node in the topology is below 5%. When controller detects it, it shutdown itself. When this happens, the script also detects the controller action and gives a commend to VM script to stop running addition to that it also stops forecast application.

The other script which runs in VM, is managing Cooja simulation. When the script receives a request to run Cooja, first it extracts the information from the request to decide which topology template is needed to run. Once, it is decided, runs the topology template on Cooja simulation from command line. To terminate the running Cooja simulation, it waits for actual script command to stop.

### 3.2.4 Test Data Reporting

To evaluate the result for each test run, there is an output file. This output file includes battery consumption, report packet delays and lifetime. These parameters are the experiments outputs that will be represented on graph in experimentation section. The output file is generated when one of the node's battery is 5% in the topology.

The battery consumption is tracked for all of the nodes until one of the node's battery reaches 5%. Once it is reached, the stored batteries consumption is written as an output. Remaining total batteries are calculated with following formula

$$255 \times n - \sum cb = \sum rb \tag{3.1}$$

$$\begin{aligned} \text{where:} \quad n &= \text{number of nodes in topology} \\ cb &= \text{consumed battery} \\ rb &= \text{remaining batteries} \end{aligned}$$

Delay is calculated using Report packets. Every 6 second, nodes are sending report packets to controller. In the payload of the report packet, last 8 byte is representing the time that is captured when the report packet is sent. The time which comes from node, is compared with the current time in the controller and the delay is estimated with this. The unit is ms for delay. Until the battery reaches 5%, delays are stored. While creating the average delay output, all existing delays are sum up and divided in the collected report packet size.

Lifetime timer starts when a first packet arrives to the controller from any node. It ends when one of the node in the topology reaches 5% battery. When the node is reached, it is written as spend time as an output to file.

There is a sample of an output file structure on Table 3.2. Type, battery weight, RSSI weight, test start time and test end time is much more informative output files for debugging. Type field could be default or forecast. The default is the behavior that runs without my changes. Forecast type is selected when battery weight is bigger than 0. Test start time is captured when the application is triggered. test end time is captured before shutting down the application.

| **Field Name** | **Value** |
|---|---|
| Type | FORECAST |
| Battery Weight | 0.5 |
| RSSI Weight | 0.5 |
| Test start time | 02019-18-16 08:18:01 |
| Test end time | 02019-18-16 08:26:55 |
| Spend time | 533s |
| Total remaining battery | 619 |
| Total battery consumption | 2186 |
| Average Delay | 430ms |

Table 3.2: Sample of Output File.

# Chapter 4

# Experiments

The experiments are conducted to compare current SDN-WISE Controller implementation and our modifications. From the results of our experiments, we have created three different evaluation graphs. While we are applying the experiment, we stick with three different weights. As shown on Table 4.1, RSSI Weight 1.0(battery is not involving) represents the current implementation of SDN-WISE Controller. RSSI Weight 0.5 (battery is involving with 50 percentage) represents that forecast involving version also battery has 50% affect with RSSI on algorithm. And last, RSSI Weight 0 (battery is only involving with 100 percentage) represents the heavily battery affect on algorithm.

From the test results, we created three graph the compare performance of weights. The graphs are lifetime, battery consumption and delay.

| RSSI Weight | Battery Weight | Short Description |
|---|---|---|
| 1 | 0 | Default behavior of SDN-WISE |
| 0.5 | 0.5 | Battery and RSSI used %50 |
| 0 | 1 | Battery usage %100 and RSSI usage %0 |

Table 4.1: Weights Used In Experiments.

|  |  |
|---|---|
| Simulation Range | Randomly distributed from 0 to 150 for X and Y |
| Number Of Sink | 1 |
| Number Of Node | 10 to 30 |

Table 4.2: Simulation Parameters.

## 4.1 Simulation Runs For Experiments

To run the experiments, ninety different simulation template has created. Each simulation template ran for RSSI Weight 1.0, RSSI weight 0.5 and RSSI weight 0.0 . The simulations topology numbers was 10, 13, 15, 18, 20, 22, 25, 28 and 30. For each topology number there was 10 different randomly created topology template. While creating the simulation template, Cooja asks you to how many number of sinks are going to be used and how many nodes are going to be used. In our experiments, sink number is 1 and nodes are changing from 10 to 30. Also, distribution of the nodes and sink is put randomly on 2D environment. X was selected as 150 and Y was selected as 150. This means that node or sink will be deployed from 0 to 150 X and 0 to 150 Y randomly. Simulation configurations are shown on Table 4.2.

To run the SDN-WISE controller, some configuration has to be set. One of the important configuration parameter is RSSI weight. If the RSSI Weight is smaller than 1. Also, SDN-WISE forecast application is being run. The possible configuration parameters are 0, 0.5 and 1. Number of nodes parameter is used for selecting query nodes randomly. To apply the query on nodes, the number of the nodes parameter value has to be known. In short, number of node and querying sending frequency from Sink parameters have a relationship between each other. Querying sending frequency from Sink parameter can not be smaller than number of nodes value. Since query nodes are always 3 and number of nodes are at least 10, there is no problem with this constraint. Report packet frequency is by default set to 6. Connection timeout is by default 25. Note that default configuration has already been used by SDN-WISE Controller. Test terminate

| | |
|---|---|
| RSSI Weight | 0, 0.5 or 1 |
| Number of Node | 10 to 30 |
| Query sending frequency from Sink | 1s for each 3 randomly selected node |
| Report packet frequency for each node | 6s |
| Connection Timeout | 25s |
| Test Terminate | When battery of a node is at 5% |

Table 4.3: SDN-WISE Controller Configuration Parameters.

parameter is newly added. The reason is to add this configuration to track the battery state of the nodes more precisely. Test Terminate is decided by setting percentage. This percentage actually represents the nodes battery. When any of the node's battery reached the percentage in topology, the SDN-WISE Controller will be terminated. The some of the SDN-WISE configuration parameters are listed in Table 4.3.

From start to end, whole experiments process is taking at least 114 hours to finish. For each weight configuration, ninety simulation is running.

Once the simulation templates are created, thanks to automation script, with one click, whole experiments are being run. The configuration parameters for Simulation and SDN-WISE Controller are set by using automation script.

The test has been triggered one time and the outputs are mapped three different graph. All these graphs are created from the test results' output files.

## 4.2 Lifetime

From the output files, we collected spend time values and created three different lines for RSSI weight 1.0, 0.5 and 0. The results are shown in Figure 4.1. Circle, triangle and star shapes represents the average of 30 different lifetime values which are the results of the different RSSI weights.

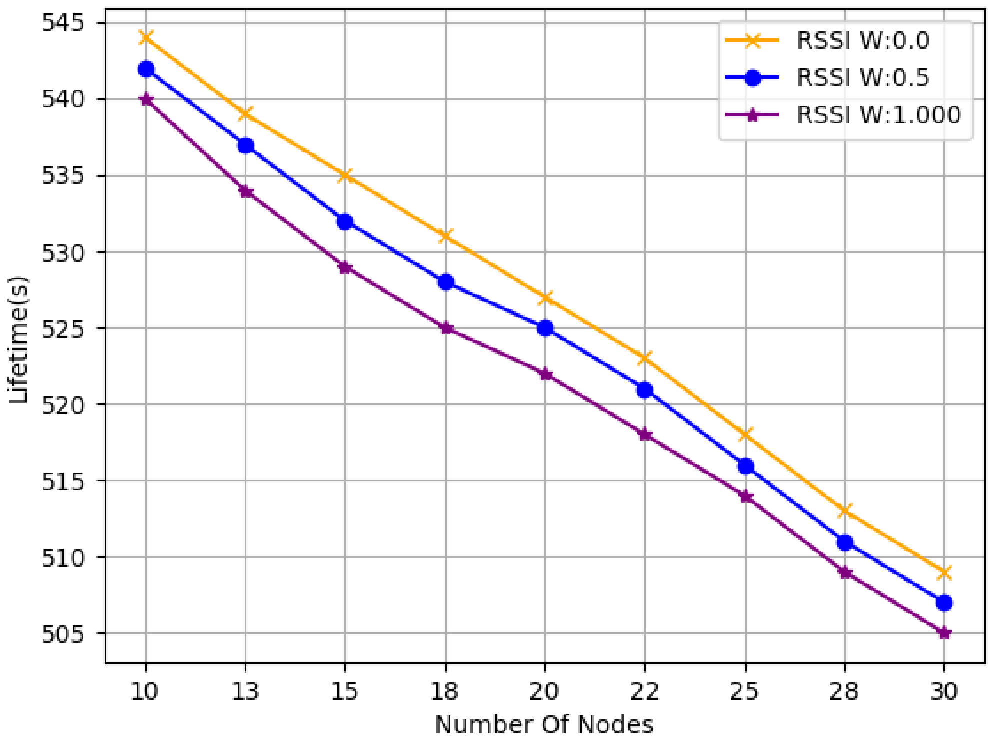

Figure 4.1: Lifetime Graph Results.

The above graph shows that we are able to reach over goal by using forecast. lifetime of the network is much increased when we set RSSI Weight value to 0. When RSSI weight is set to 0.5, we observe that the lifetime increase as well. In addition to that, we can observe that the lifetime is getting higher when we decrease the RSSI weight. With 30 sample there is no intersection point between lines. As last, from the graph we can see that even RSSI Weight 0.5 is not going below of the RSSI weight 1.0 or above RSSI weight 0.

## 4.3 Battery Consumption

In the output files, we also collected battery consumption values and created three different lines for RSSI weight 1.0, 0.5 and 0. The battery consumption results are shown in Figure 4.2. Again, circle, triangle and star shapes represents the

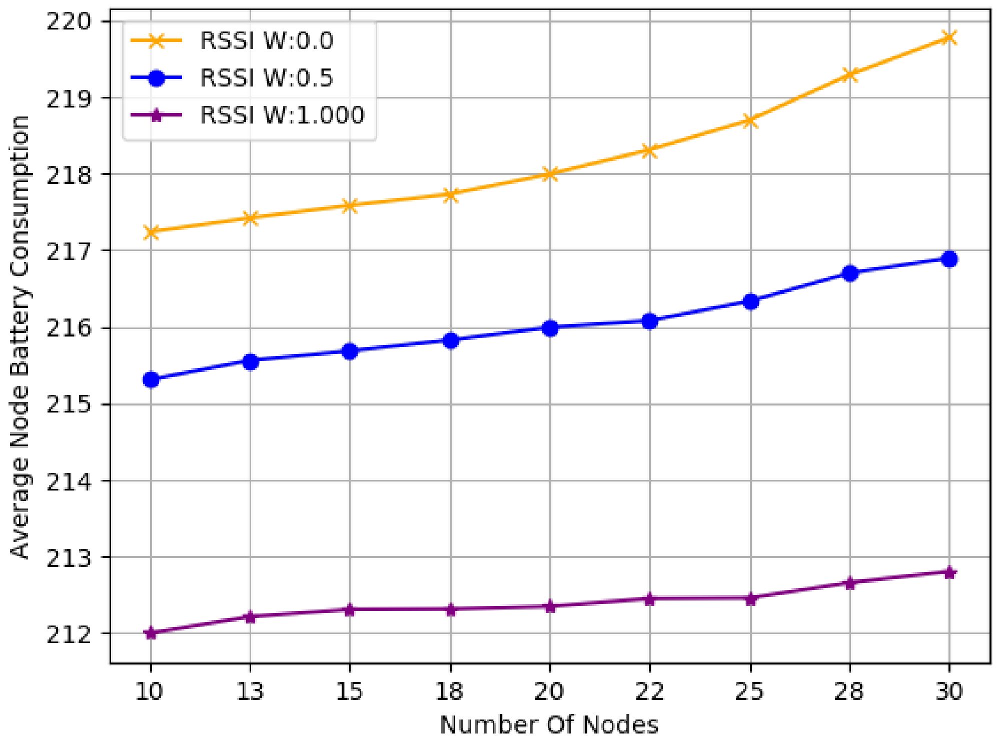

Figure 4.2: Battery Consumption Results.

average of 30 different battery consumption values which are the results of the different RSSI weights.

The above graphs shows that when we include the battery in Dijkstra algorithm, the battery consumption is going to be higher compared to RSSI Weight 1.0. Also, in the graph, we can see that RSSI Weight 1.0 is less consumed battery compared to 0 and 0.5. The reason is the involvement of the battery. It is causing routing, that is why it is going different paths. Using different path is causing more battery consumption. Moreover, when RSSI Weight 1.0, the line is nearly linear because battery is not involving in the algorithm.

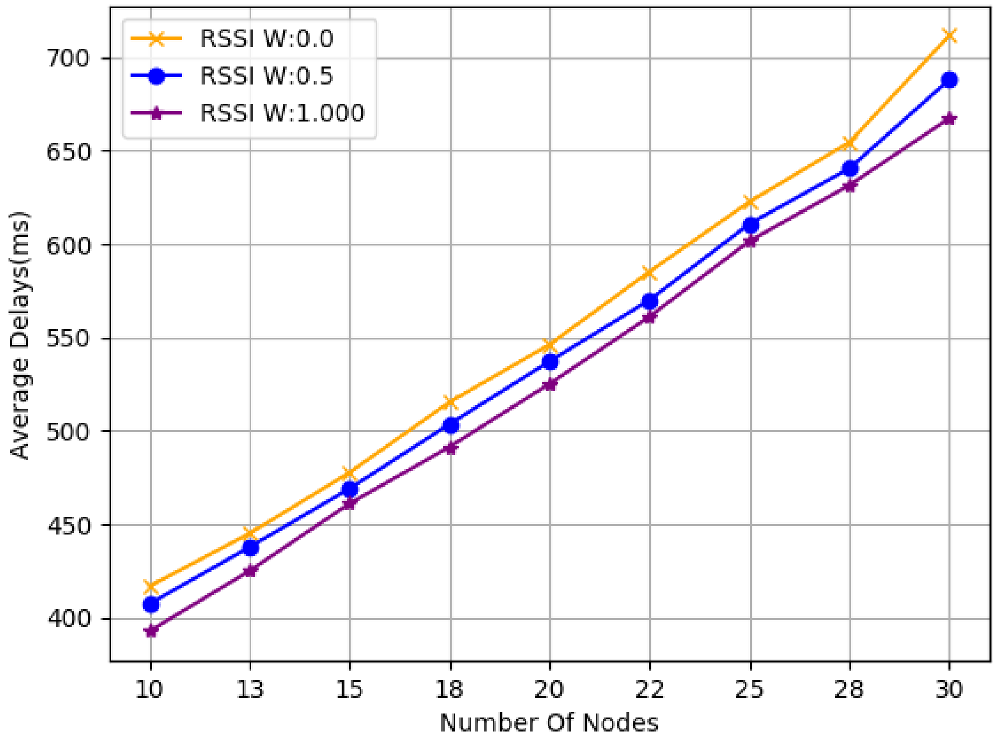

Figure 4.3: Delay Results.

## 4.4 Delay

As last from the output files, we collected average delays for report packets and created three different lines for RSSI weight 1.0, 0.5 and 0. The average delay results are shown in Figure 4.3. Again, circle, triangle and star shapes represents the average of 30 different average delay values which are the results of the different RSSI weights.

The above graphs shows that even we increase lifetime from the forecast, we increase the delay as well. Increasing the delay is reducing the performance of the network. The reason for increasing delay could be the number of rerouting. Since battery is changed more frequently, the rerouting occurs more frequently. Routing triggers the topology discovery packets to propagate through the network for more efficient rerouting. In that time, nodes are being in preparing state.

Thus, there could be reason that some of the nodes are not able to send their report packets to controller due to this rerouting. We can observe that RSSI Weight 0 delays are higher compared to others, and RSSI weight 0.5 is higher that RSSI weight 1.0. From this result, we can understand that adding battery to algorithm, is causing overhead for routing.

# Chapter 5

# Conclusion

In this study, we propose a solution to increase the lifetime of the wireless sensor network using battery forecasting. The lifetime and battery consumption graphs show that we accomplished our objective and able to increase lifetime of the wireless sensor network. However, the delay graph shows that when we try to reach our goal, we did trade-off unintentionally. While we were increasing the lifetime of the network, we also increased data transfer the delay between the nodes. The situation might be occurred due to frequent rerouting or SDN-WISE topology discovery algorithm which might not be not good fit for applying forecast. In addition to that, the feature set might be small, we could find another features to solve this issue while we are using the linear regression. In this solution, we just prefer to use time stamp as a feature in linear regression. Moreover, another reason might be the our forecast approach. While we are doing that we preferred the linear regression instead of different machine learning methods. Instead of using linear regression, we could use another ML algorithm. As an example, Recurrent Neural Network (RNN). It is a time series machine learning algorithm that might work efficiently with this solution. However, because of the time limitation, there wasn't much time to learn new machine learning approach. Also, existing code base was not ready to integrate machine learning. Thus, understanding the RNN, could take much time. Moreover, we needed to investigate how it is going to behave with different kind of WSNs. As a future work, it could be used for this proposed solution instead of linear regression.

Test automation was the important part of this study. In existing tutorial and papers, the way of testing is not explained in details. In this study, we also focused on testing some chapters. We tried to explain the details of testing and support our explanation with figures.

# References

[1] K. M. Modieginyane, B. B. Letswamotse, R. Malekian, and A. M. Abu-Mahfouz, “Software defined wireless sensor networks application opportunities for efficient network management,” *Comput. Electr. Eng.*, vol. 66, no. C, pp. 274–287, Feb. 2018. [Online]. Available: https://doi.org/10.1016/j.compeleceng.2017.02.026

[2] M. Jacobsson and C. Orfanidis, *Using software-defined networking principles for wireless sensor networks*, 2015. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-254172

[3] D. Duan, F. Qu, W. Zhang, and L. Yang, “Optimizing the battery energy efficiency in wireless sensor networks,” in *2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, Sep. 2011, pp. 1–6.

[4] Jian-Quan Wang, Haijing Fu, and Chang Cao, “Software defined networking for telecom operators: Architecture and applications,” in *2013 8th International Conference on Communications and Networking in China (CHINACOM)*, Aug 2013, pp. 828–833.

[5] SDN-WISE. SDN-WISE tutorial with vm. Accessed: 24 December 2018. [Online]. Available: https://sdnwiselab.github.io/docs/guides/GetStarted.html

[6] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, “Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor

networks," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 513–521.

[7] L. Galluccio, S. Palazzo, S. Milardo, and G. Morabito, "Reprogramming wireless sensor networks by using sdn-wise: A hands-on demo," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2015, pp. 19–20.

[8] A. G. Anadiotis, G. Morabito, and S. Palazzo, "An sdn-assisted framework for optimal deployment of mapreduce functions in wsns," *IEEE Transactions on Mobile Computing*, vol. 15, no. 9, pp. 2165–2178, Sep. 2016.

[9] N. Q. Hieu, N. Huu Thanh, T. T. Huong, N. Quynh Thu, and H. V. Quang, "Integrating trickle timing in software defined wsns for energy efficiency," in *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*, July 2018, pp. 75–80.

[10] S. Milardo, A. Venkatasubramanian, K. Vijayan, P. Nagaradjane, and G. Morabito, "From reactive to predictive flow instantiation: An artificial neural network approach to the sd-iot," in *European Wireless 2018; 24th European Wireless Conference*, May 2018, pp. 1–6.

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

[12] M. S. Olimjonovich, "Software defined networking: Management of network resources and data flow," in *2016 International Conference on Information Science and Communications Technologies (ICISCT)*, Nov 2016, pp. 1–3.

[13] A.-C. Anadiotis, L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "Sd-wise: A software-defined wireless sensor network," *Computer Networks*, vol. 159, pp. 84 – 95, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128618312192

[14] SDN-WISE. SDN-WISE protocol. Accessed: 24 December 2018. [Online]. Available: https://sdnwiselab.github.io

[15] I. Community. InfluxDB documentation. Accessed: 15 January 2019. [Online]. Available: https://docs.influxdata.com/influxdb/v1.6/

[16] G. Community. Grafana documentation. Accessed: 02 January 2019. [Online]. Available: https://grafana.com/grafana

[17] D. Community. Docker documentation. Accessed: 05 June 2019. [Online]. Available: https://www.docker.com/resources/what-container

[18] Sajari. Sajari regression library. Accessed: 26 December 2018. [Online]. Available: https://github.com/sajari/regression

[19] WMWare. WMWare documenntation. Accessed: 30 May 2019. [Online]. Available: https://docs.vmware.com/en/VMware-Workstation-Player/index.html

[20] A. Dunkels. Contiki documentation. Accessed: 24 December 2018. [Online]. Available: https://github.com/contiki-ng/contiki-ng/wiki

[21] C. Maintainers. Introduction to Cooja. Accessed: 24 December 2018. [Online]. Available: https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja

[22] U. Community. Ubuntu releases. Accessed: 24 March 2019. [Online]. Available: https://wiki.ubuntu.com/Releases