

**T.C.
SAKARYA UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY**

SDN INTEGRATION FOR INTERNET OF THINGS USING WSN AND RFID

Ph.D. THESIS

Mohammed Hussein Al-HUBAISHI

**Department : COMPUTER AND INFORMATION
ENGINEERING**
Supervisor : Prof. Dr. Celal ÇEKEN

November 2019

**T.C.
SAKARYA UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY**

**SDN INTEGRATION FOR INTERNET OF THINGS
USING WSN AND RFID**


Ph.D. THESIS


Mohammed Hussein Al-HUBAISHI


**Department : COMPUTER AND INFORMATION
ENGINEERING**
Supervisor : Prof. Dr. Celal ÇEKEN


**This thesis has been accepted unanimously / with majority of votes by the
examination committee on 29/11 /2019**


Prof. Dr. Celal Çeken
Head of Jury


**Prof. Dr. Emine Doğru
Bolat**
Jury Member


Asst. Prof. Dr. Seçkin Arı
Jury Member


Assoc. Prof. Dr. Ali Çalhan
Jury Member


Assoc. Prof. Dr. Devrim Akgün
Jury Member

DECLARATION

I declare that all the data in this thesis was obtained by myself in academic rules, all visual and written information and results were presented in accordance with academic and ethical rules, there is no distortion in the presented data, in case of utilizing other people's works they were refereed properly to scientific norms, the data presented in this thesis has not been used in any other thesis in this university or in any other university.

Mohammed Hussein Al-Hubaishi
29/11/2019



ACKNOWLEDGEMENT

Firstly, thank you, Almighty Allah, for giving me enough strength to perform and complete my Ph.D. thesis study successfully.

Secondly, I would like to give credit with deep appreciation and respect to my advisor Prof. Dr. Celal Çeken for his valuable efforts, unprecedented support and priceless advice since the first day we met. Also, I would like to express my thanks to the jury members.

Furthermore, I would like to extend my thanks to YTB (Yurtdışı Türkler ve Akraba Topluluklar Başkanlığı) for financial support during my Ph.D. study. Also, I would like to acknowledge that this work was supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) with project number (116E008). Likewise, to my friends in Sakarya, and my colleagues in the IoT Research Laboratory, many thanks for you, especially to Ali Burhan and Nur Banu OĞUR for their work with us in the projects.

Finally, I deeply thank all my family members especially my lovely mother, father, my wife and my children for their unconditional love, valuable advice, encouragement, trust, prayers, for giving me a reason to keep moving forward. Thank you, my friends, in Sakarya and Sakarya University for the happy time they have shared with me.

TABLE OF CONTENTS

ACKNOWLEDGEMENT.....	i
TABLE OF CONTENTS	ii
LIST OF SYMBOLS AND ABBREVIATIONS	v
LIST OF FIGURES	vi
LIST OF TABLES	viii
SUMMARY	ix
ÖZET	x
CHAPTER 1.	
INTRODUCTION	1
1.1. Motivation and Problem Statement.....	1
1.1.1. Motivation.....	1
1.1.2. Problem statement.....	2
1.2. Thesis Goals.....	3
1.3. Research Contributions.....	3
1.4. Related Work.....	4
1.4.1. WSN routing.....	4
1.4.2. SDN-based WSN routing protocols.....	6
1.5. Thesis Structure.....	7
CHAPTER 2.	
THE INTEGRATION OF SDN-WSAN FOR INTERNET OF THINGS USING IEEE 802.15.4.....	8
2.1. Introduction.....	8
2.2. SDN-based WSAN for Internet of Things.....	9
2.3. Software-Defined Networking Architecture for IoT.....	11
2.3.1. SDN application plane.....	12

2.3.2. SDN control plane.....	13
2.3.3. SDN data plane.....	13
2.3.4. OpenFlow (WSANFlow) protocol stack.....	16
CHAPTER 3.	
THE SDNC FUZZY-BASED ROUTING DECISION	17
3.1. Introduction.....	17
3.2. The SDNC Fuzzy-based Routing Decision.....	18
3.3. Combined Flow Tables.....	21
3.4. Network Topology.....	22
3.5. Node Status Aggregation (NSA)	23
3.6. Fuzzy Logic.....	24
3.6.1. Fuzzification.....	25
3.6.2. Rule evaluation.....	27
3.6.3. Defuzzification.....	29
3.7. Dijkstra's Algorithm.....	32
3.7.1. Flow tables layout.....	34
3.8. Simulation Results and Discussion.....	36
3.8.1. Simulation environment: parameters and scenarios.....	36
3.8.2. Simulation results and discussion.....	37
CHAPTER 4.	
INTERACTING THE VERTICAL DOMAIN DEVICES OF IOT SYSTEM WITH HORIZONTAL DOMAIN SYSTEM	43
4.1. IoT Technologies Related Work.....	43
4.2. Vertical Domain Devices	45
4.2.1. Software-defined networking based WSN.....	47
4.2.2. Radio frequency identification.....	48
4.3. Horizontal Domain System.....	50
4.3.1. Kafka.....	50
4.3.2. Apache spark.....	54
4.4. Case Study: Real-Time Disease Diagnosis by Logistic Regression	54

4.4.1. Electrocardiogram.....	54
4.5. Experimental Results and Discussions.....	55
4.5.1. Simulation environment and hypothesis.....	57
CHAPTER 5.	
ENHANCED SDN TOPOLOGY DISCOVERY AND IOT GATEWAY	63
5.1. Introduction.....	63
5.1.1. Related research.....	63
5.2. The Proposed ESWFD for IoT Applications.....	64
5.2.1. Components of the ESWFD system.....	65
5.2.2. SDN Topology Discovery	66
5.2.3. Message queue telemetry transport.....	68
5.2.4. The ESWFD system operations.....	70
5.3. Experimental Results and Discussions.....	71
5.3.1. Simulation scenario and discussion.....	72
5.3.2. Performance analysis and scenario results.....	73
CHAPTER 6.	
OVERALL CONCLUSION AND FUTURE WORK.....	76
REFERENCES.....	79
RESUME.....	86

LIST OF SYMBOLS AND ABBREVIATIONS

SDN	: Software Defined Networking
IoT	: Internet of Things
MQTT	: Message Queue Telemetry Transport
WSN	: Wireless Sensor Networks
WSAN	: Wireless Sensor and Actuator Network
ESWFD	: Enhanced SDN-WSAN Fuzzy-based Dijkstra
SD	: Superframe Duration
FDB	: Fuzzy-based Dijkstra's Beacon
FD	: Fuzzy-based Dijkstra's
RDFT	: Routing Decision and Flow Table
RFID	: Radio frequency identification
SNR	: Signal-to-Noise Ratio
BI	: Beacon interval

LIST OF FIGURES

Figure 1.1. Routing Energy Problem in WSN	3
Figure 2.1. The Architecture of the SDN-based WSN clusters for IoT	12
Figure 2.2. IEEE 802.15.4 superframe [43]	15
Figure 2.3. Superframe inactive period (sleep time).....	16
Figure 3.1. Architecture design of routing protocol in the SDN controller	19
Figure 3.2. Sequence diagram of the system processes	20
Figure 3.3. Combined flow table class.....	21
Figure 3.4. The network topology.....	23
Figure 3.5. Node status aggregation	24
Figure 3.6. Fuzzy logic with inputs and output.....	25
Figure 3.7. Fuzzy inputs SNR and battery level memberships.....	25
Figure 3.8. SNR, battery membership functions map the input values	26
Figure 3.9. Rule evaluation results	28
Figure 3.10. Fuzzy output cost membership	29
Figure 3.11. Example fuzzy results.....	30
Figure 3.12. The flowchart of the defuzzification procedures	31
Figure 3.13. Dijkstra's with fuzzy routing algorithm.....	33
Figure 3.14. Dijkstra's algorithm.....	34
Figure 3.15. Network topology Scenario 1	35
Figure 3.16. Network topology Scenario 2	35
Figure 3.17. Network topology Scenario 3	36
Figure 3.18. First ED deaths with number of EDs and number of applications	38
Figure 3.19. Scenario for 50 EDs and one application	39
Figure 3.20. Number of ED deaths over time	40
Figure 3.21. Network loads for the intermediate end devices	41
Figure 3.22. Throughput (bit/sec) and end-to-end delay	41

Figure 4.1. Outline of the RFID in vertical domain architecture.	47
Figure 4.2. SN with RFID reader node model and RFID reader process model.	48
Figure 4.3. RFID tag node model and process model.	49
Figure 4.4. Streaming gateway and Kafka (producer/consumer).	51
Figure 4.5. The commands to run zookeeper and Kafka servers	52
Figure 4.6. Include the c-based socket in riverbed node model.	52
Figure 4.7. The destination SN (gateway) use the socket to send packet to Kafka ...	53
Figure 4.8. Java based socket.	53
Figure 4.9. The Sequence diagram for end-to-end data delivery operations.	55
Figure 4.10. NodeJS patient information interface.	57
Figure 4.11. Streaming scenario for IoT healthcare applications.	58
Figure 4.12. Logistic regression predicted results.	60
Figure 4.13. The end to end delay for the three applications.	62
Figure 4.14. Throughput (bit/s) for the three applications.	62
Figure 5.1. The general description of the ESWFD proposed IoT architecture.	65
Figure 5.2. The system components, the node model of SDN (a) and the node model of sensor node (b).	66
Figure 5.3. Flowchart of two algorithms to calculate the SD, Fuzzy-based Dijkstra (A) and Fuzzy-based Dijkstra Beacon (B).	67
Figure 5.4. MQTT broker and client communication.	69
Figure 5.5. The sequence diagram for end-to-end data delivery operations.	70
Figure 5.6. NodeJS information interface.	72
Figure 5.7. The simulation model of the SDN controller and 50 sensor nodes	73
Figure 5.8. Throughput (bit/s) for 50 nodes.	74
Figure 5.9. End-to-End Delay for 50 nodes	74
Figure 5.10. Jitter for 50 nodes	75
Figure 5.11. Delay, Jitter and Throughput on different network sizes	75

LIST OF TABLES

Table 2.1. Beacon interval and superframe duration evaluation	15
Table 3.1. Fuzzy logic rule base	28
Table 3.2. Cost vs battery level.....	31
Table 3.3. Flow table entries	34
Table 3.4. Simulation parameters	37
Table 3.5. Path information	39
Table 4.1. Logistic regression algorithm parameters.	59
Table 4.2. Classification table.....	59
Table 4.3. Data streaming.	59
Table 4.4. Total average of the input rate.	60
Table 4.5. Streaming Data with Number of Partitions.....	61
Table 5.1. Algorithms Configurations	68
Table 5.2. Parameters of the Simulation	73

SDN INTEGRATION FOR INTERNET OF THINGS USING WSN AND RFID

SUMMARY

Keywords: Software-Defined Networking, SDN, Wireless Sensor Networks, WSN, Internet of Things, IoT, RFID, Apache Kafka, Apache Spark.

Internet of Things (IoT) has started to touch every aspect of our lives, from home automation, smart factories, energy management systems, precision agriculture to smart city systems, etc. Wireless Sensor Networks (WSN) play an important role in various IoT applications. One of the challenging problems for any WSN design is lack of flexibility in network management. Software-Defined Networking (SDN) is a new approach that promises a more flexible and dynamically reconfigurable network structures. When designing WSN, energy problem must also be considered since each device in the network has limited battery capacity.

This study proposes a new routing discovery algorithm which allows the SDN-enabled WSN to make smarter routing decisions considering the received signal strength and remaining energy of the devices. The new architecture employs a fuzzy-based Dijkstra's algorithm when deciding the best path between the source and the destination. The study also introduces a new real-time data analytics architecture for IoT applications, consisting of a WSN and radio frequency identification (RFID) technology in the vertical domain. The platform proposed also has highly scalable and high-performance data analytics tools such as Apache Kafka, Apache Spark and MongoDB, in the horizontal domain.

Simulation results show that the proposed routing discovery mechanism can provide effective clustering routing for SDN-based WSN and can prolong the network lifetime by reducing the energy consumption of the nodes in the network. The results also show that the proposed IoT data analytics system can process data in real-time, successfully, and is capable of handling large amounts of data easily, owing to the scalable technologies deployed.

ÖZET

Anahtar Kelimeler: Yazılım Tanımlı Ağlar, Kablosuz Algılayıcı Ağlar, Nesnelerin İnterneti, Radyo Frekanslı Tanımlama, Apache Kafka, Apache Spark.

Nesnelerin İnterneti (Nİ) ev otomasyonundan akıllı fabrikalara, enerji yönetim sistemlerine, hassas tarımdan akıllı şehir sistemlerine vb. kadar hayatımızın her alanına dokunmaya başladı. Kablosuz Algılayıcı Ağlar (KAA) çeşitli Nİ uygulamalarında önemli bir rol oynamaktadır. KAA tasarımları için en önemli sorunlardan biri, ağ yönetiminin oldukça zor olmasıdır. Yazılım Tanımlı Ağlar (YTA), daha esnek ve dinamik olarak yeniden yapılandırılabilir bir ağ yapıları vaat eden yeni bir yaklaşımdır. KAA'ı tasarlarırken, ağdaki her cihazın sınırlı pil kapasitesi olduğundan, enerji sorunu da göz önünde bulundurulması gereken önemli sorunlardan biridir.

Bu çalışma, YTA özellikli KAA'nın alınan sinyal gücü ve cihazların kalan enerjisi dikkate alınarak daha akıllı yönlendirme kararları vermesini sağlayan yeni bir yönlendirme keşif algoritması önermektedir. Yeni mimari, kaynak ve hedef arasındaki en iyi yolu belirlerken bulanık tabanlı bir Dijkstra algoritmasını kullanır. Çalışma ayrıca, dikey alanında KAA ve radyo frekanslı tanımlama (RFID) teknolojileri içeren Nİ uygulamaları için yeni bir gerçek zamanlı veri analitiği mimarisi de içermektedir. Önerilen platform yatay alanda Apache Kafka, Apache Spark ve MongoDB gibi yüksek düzeyde ölçeklenebilir ve yüksek performanslı veri analizi araçlarına sahiptir.

Benzetim sonuçları, önerilen yönlendirme keşif mekanizmasının YTA tabanlı KAA için etkili kümelenme yönlendirmesi sağlayabildiğini ve ağdaki düğümlerin enerji tüketimini azaltarak ağ ömrünü uzatabileceğini göstermektedir. Sonuçlar ayrıca önerilen Nİ veri analitiği sisteminin verileri gerçek zamanlı olarak başarılı bir şekilde işleyebildiğini ve konuşlandırılan ölçeklendirilebilir teknolojiler sayesinde büyük miktardaki verileri kolayca ele alabildiğini göstermektedir.

CHAPTER 1. INTRODUCTION

This chapter provides an introduction containing a motivation and problem statement, the goal of the research, research contributions, and the related work to the thesis. In this chapter, the problem statement identifies the main issue that this research attempts to address.

1.1. Motivation and Problem Statement

1.1.1. Motivation

The Internet of Things (IoT) has increased the number of devices on the internet, largely because of the need to obtain information from these devices for various IoT applications, so that the number of these devices is expected to continue to rise [1]. Cisco [2] expects that the number of Internet-connected devices will rise to approximately 50 billion by next year.

Nowadays, wireless sensor and actuator networks (WSANs) [3][4] play a significant role in data transfer from several systems through their various applications. Applications include environmental monitoring data such as air temperature, humidity, smog-like gasses and precision agriculture [5]. WSAN networks have become very important and preferred for several applications due to characteristics such as their low cost, small size, low power consumption, mobility, and multifunctional sensors. For these reasons, integration of WSAN and IoT networks is useful for IoT applications, but there are also challenges to meet IoT application requirements such as energy efficiency, flexible management, and network reconfiguration after deployment [6][7].

The most important design constraints when constructing a WSN is energy efficiency, because each device operates with limited power resources. These devices consume more power when they send or receive data depending on the routing protocol used [8][9]. Therefore, the routing mechanism has to take in energy consumption parameters to be controlled. Thus, a smart and flexible network is needed for forwarding data among those devices.

Moreover, the routing protocol must be designed in such a way as to maximise the lifetime of the network by saving energy [10][11]. The routing decision used in traditional wireless network sensors [4] does not take into account energy consumption during communication, which results in unmeasured energy consumption, unwanted delays and a great deal of overhead traffic. These results are required for some IoT applications, since IoT applications' performance relies on the forwarding and routing methods. Therefore, IoT applications need to make appropriate decisions based on the lowest possible path cost to send data through WSN nodes from the source to the target. This challenge requires an intelligent network energy administration for WSN devices.

Network management and control design form the key part of the solution, the software-defined networking (SDN) architecture [12], which aims to provide solutions such as flexible management and network reconfiguration after deployment to WSN. Since management of the limited battery is the most important task of a WSN, as we will see in the literature, there are many different proposals for this purpose.

The research interest of this thesis is to integrate the concept of SDN for IoT technologies and perform better flexible network energy management for WSN devices.

1.1.2. Problem statement

The integration of the SDN-based WSN for IoT, is expected to improve the energy management of WSN via a flexible and programmable network and to support IoT

applications requirements [11-40]. However, based on the literature survey aimed at establishing a centralised network of sensor networks, there are two main problems with the implementation of SDN-based WSN, as shown in Figure 1.1.: efficient routing of energy for WSN devices, and the integration of IoT technologies. The purpose of this study is to overcome these problems and propose a new structure to improve system performance.

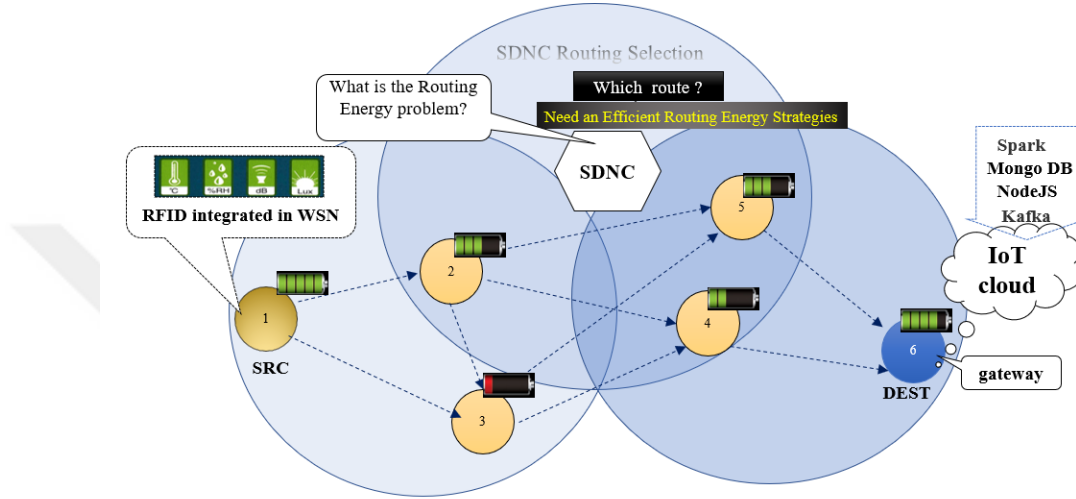


Figure 1.1. Routing Energy Problem in WSN

1.2. Thesis Goals

The research goal of this thesis is to develop a new energy-aware routing mechanism in the SDN controller and to combine IoT technologies into the system, so that the SDN controller can act as a smart and flexible network energy manager using a new routing discovery mechanism for WSN devices. In addition, to integrate IoT technology – namely radio frequency identification (RFID) – into the WSN source devices as part of vertical domain devices. Furthermore, the communication and implementation between the vertical and horizontal domain devices will be included in the research.

1.3. Research Contributions

The most important component in the SDN controller is the routing decision model. In this study, we use Riverbed Modeler as a practical performance evaluation for better

performance evaluation. The main contributions of this research can be summarised as follows:

- A simple and flexible network structure and management was developed to reduce energy consumption for SDN-based WSN architectures.
- An SDN-inspired approach, a new framework for WSN, was designed.
- An RFID model integrated with WSN was developed.
- Node status aggregation and fuzzy-based Dijkstra's algorithms were developed in the SDN controller to take a routing decision approach to maximising the network lifetime.
- A new real-time IoT-based data analytics architecture for smart healthcare is proposed. This new platform has an SDN-based WSN and an RFID structure, in the vertical domain.
- To enhance the performance of the system, a new routing discovery strategy via cross-layer (Topology Discovery and MAC layers) was developed.
- To link IoT vertical and horizontal systems, two different gateways were developed, one using TCP/IP socket client/server programs and the other using MQTT broker/client, allowing the system to send WSN data to the IoT cloud.
- An IoT web interface using NodeJS was developed for monitoring network configuration from the SDN controller and data of the IoT application from destination node.

1.4. Related Work

In the following literature review, we discuss the current work on WSN routing protocols and SDN-based WSN routing protocols.

1.4.1. WSN routing

Traditional wireless sensor networks are widely used by many applications in different areas. A survey on WSN applications and energy is given in [9], in which WSN applications were distributed into five major groups: node deployment, targets, service area, required measurements, and data transmission requirements. The routing

protocol applications for WSN in [13] are classified into three groups: environment-specific, task-specific, and general. Routing challenges are presented in [14] [15] for WSNs that affect routing in the matter of power consumption, sensor deployment, link diversity, adaptability, fault sufferance, broadcasting, communication, encasement, quality of service and data collection.

Although many investigations have been conducted into WSN routing protocols, they are still un-reprogrammable and unconfigurable during real-time operations for new variables in the network to keep up with the applications requirements. Below, we summarise the surveys performed on the routing protocols designed for WSN with regard to network topology information, divided into three groups: hierarchical, flat, and location-based routing.

Hierarchical routing: In this type of protocol, which is a cluster-based routing protocol, the nodes are divided into parent node as cluster head (CH) and child node as sensor node. The CH behaves as a router to decide on routes in order to pick the best route. The network is also divided into several clusters, so that the parent node aggregates the information of its child nodes. It uses the higher energy to process and send the information [15]. The CH coordinate is selected between the clusters and the routing activities and transmits information among the cluster nodes. Child nodes use the low energy to send their status information to the parent node and subsequently turn off to save energy. The CH collects the data and transmits it to the base station, which means that the CH reduces the number of packets sent to the sink or reduces the number of packets from a lower clustered layer to a higher one. This helps to reduce power consumption in the network [16] [17]. While sensor nodes have different roles in hierarchical routing, clustering forwarding is an efficient way to lower power consumption and extends the network lifetime [8] [18].

Flat-based routing: In this type of protocol, each node at the same level acts as a router for forwarding data, so that the same role is shared by, and the sensing task is equally divided among all sensor nodes [15]. The routing procedures for each node lead to more power consumption. This method is good for maintaining network problems.

Flat-based routing uses the flooding packets to forward data toward the destination node [16]. Therefore, it also suffers from packet overhead between communicating sensor nodes, and limited scalability. In this approach, the base station sends queries to a certain node's area, and the node that received the query information will relay to the base station in that area [18] [8].

Location-based routing: In these protocols, the routing mechanism depends on the location of the sensor nodes. During the broadcast processes, the incoming signal strengths are used to estimate the distances between neighbouring nodes. This method is considered to be very useful in applications that require the location of sensor nodes to route their data. Some location-based schemes for the sensor nodes that do not have any activities in the network are asked to sleep to save energy [15], the energy consumption having been calculated using the sensor node's location information [16]. Location-based routing reduces the overhead involved in finding the route to base station and increases scalability.

1.4.2. SDN-based WSN Routing Protocols

Recently, the possibility has opened up of reprogramming networks depending on the SDN concept [19]. Here, we present some research which discusses routing protocols in SDN-based WSN where the routing is based on flow table.

In the traditional SDN approach, the article [20] proposes a hierarchy-based architecture routing protocol to solve control plane scalability for possible paths between source to destination sensor nodes; the concept of SDN follows the hierarchical architecture in terms of an inter-AS routing approach.

The authors of [21] propose an SDN-enabled WSN architecture where the SDN controller runs the SPIN routing protocol using the Cooja simulator. The SPIN protocol generates a neighbours table, and the SDN selects the best of these neighbours based on its voltage level and link quality to forward the sensor data. Differences of

routing approaches between traditional wired SDN and SDN-based wireless sensor network can be found in [22].

SDN-ECKKN is proposed in [11], where the routing method includes the time interval of each node, divided into beacon and execution slices to save energy. In this study, there is no broadcasting between each pair of nodes.

The SensorSDN framework in [23] proposes a new control layer service to support topology discovery and management of network policies. For routing purposes, the packets pass from the network layer to the MAC layer and the latest matching rules based on the technologies of LR-WPAN are introduced for SensorSDN.

μ SDN proposes in [24] to extend the current AODV and LARP routing protocols to a centralised routing method based on the SDN. This work modifies the current routing messages and identifies new types of message for the SDN controller to perform routing with.

The authors in [25] propose FJAPSO to offer a green routing algorithm for SDN-based WSN. The SDN controller uses FJAPSO to select the parent nodes as cluster heads to perform routing processes. The FJAPSO's selection procedure is performed during the iteration loop: by using a mutation operator, each particle is subdivided into multiple sub-particles and subsequently incorporated into a root particle based on their fitness with the best sub-particle patterns. FJAPSO works on two levels for automatic optimisation, optimal number and optimal cluster of the control nodes.

1.5. Thesis Structure

In Chapter 2, we discuss the SDN controller architecture and SDN sensors in detail. We also discuss the SDNC fuzzy-based routing decision in Chapter 3. Chapter 4 discuss the integration of the RFID model into the sensor model. Chapter 5 covers the enhanced SDNC topology discovery model used in this thesis. Finally, Chapter 6 concludes the thesis.

CHAPTER 2. THE INTEGRATION OF SDN-WSAN FOR INTERNET OF THINGS USING IEEE 802.15.4

2.1. Introduction

In the last few years, software-defined networking (SDN) has been a growing interest within computer network communities as a new solution for reconfiguring and controlling a network according to application requirements [26][27]. In the SDN architecture, which aims to separate data and control planes [28], all the control functions of the network are performed by a centralised device referred to as the SDN controller (SDNC). On the other hand, network devices in the data plane are responsible only for forwarding the data using the related entry in their flow tables. The advantage of using the SDNC is that the applications' data relies on the forwarding and routing method's performance. Besides that, the SDNC device should make an appropriate decision based on the lowest cost path from the source to the destination, since paths with higher costs make devices consume a great deal of energy in a short time.

Software-defined networking developed as an intelligent solution to many wired network issues: troubleshooting, traffic management, and resources on the network deployed in cloud data centres [29][23]. Despite these developments, SDN is creating tremendous enhancements in network programming that do not require any hardware replacement.

Although the SDN concept was built on wired networks, it was not planned to implement the SDN concept in a WSAN. Many researchers have recently discussed the issues with integrating SDN into a WSAN network structure.

On one hand, WSNs contain a variety of applications that have become essential parts of our lives [9]. On the other hand, the IoT has greatly expanded the number of application areas, particularly in the smart home, healthcare, transportation, surveillance, security, and other industrial necessities [30][31]. These applications are deployed in numerous environments and areas, so that their requirements are various, enlarging the management challenges for the network and devices.

The integration of SDN-based WSNs poses many challenges, most importantly energy routing management [32] [33]. The WSN devices can range from the edge for the IoT cloud, actuator devices and wireless sensors. The IoT edge is used to collect data in order to send it to the cloud, which contains tools to convert this data into meaningful information [30]. The IoT cloud requires a network to support real-time data transfer in order to analyse the data for some IoT applications. Therefore, an effective data routing strategy with low power consumption for WSNs should be smart and flexible in the SDN controller to meet IoT requirements.

In this chapter, we discuss the proposed SDN-based WSN for IoT. The SDN controller, sensor node and SDN integration for IoT are described in detail in the following sections.

2.2. SDN-based WSN for Internet of Things (IoT)

There are many challenges involved in gathering extensive data in real time from IoT technologies in order to analyse this data and make appropriate decisions on a large scale. Software-defined networking is proposed as a flexible network architecture to face these challenges for IoT requirements.

The authors in [26] discuss the challenges of SDN integration with IoT in terms of security and scalability. SDN-SPS [34] designed a gateway node (SITL) to transfer the data from OPNET simulator nodes to an IoT data centre using UDP protocol. In this work, the SDN controls the time, frequency and data types.

The routing method of RPL and TinySDN protocols in [35] uses a collection tree protocol to perform routing and operates based on DODAG control messages. RPL works to improve TinySDN by supporting point-to-point and point-to-multipoint traffic patterns for IoT. Each child node determines its parent node to find a way to the coordinator node to forward its packets, and subsequently will not send DIO messages. The SDN model is used in this work to improve WSN and IoT deployment through its flexible management and control on the sharing resource.

The paper in [36] presented SDN controller for WSNs as a floodlight controller in the Mininet network emulator tool. The sensor nodes of the WSNs are implemented in NS2, another simulation network, and both systems are connected via port-to-port communication. In this research, the packet routing processes are achieved through multi pipeline stages; for routing commands, it uses the command line interface to pass the flow rules.

The SDN/NFV model in [37] used Mininet and POX controller to support the services of IoT applications, such as smart home, self-driving cars, and e-healthcare. The IoT services were hosted on HomeGWs as network gateways to decrease core network traffic. The HomeGWs played important roles for routing purposes, forwarding packets to the destination.

The S-MANAGE protocol is proposed in [38]; it features an interface to control the management messages between the SDN controller and the virtual IoT sensors. The routing operations in this protocol are underpinned by OpenFlow and OF-CONFIG.

Soft-WSN in [39] is a proposed integration of device and topology managements into SDN controller for managing IoT devices based on the IEEE 802.15.4 and IEEE 802.11 protocols. The device management is responsible for active-sleep scheduling, sensing tasks, and sensing delay. The topology management module is used for routing purposes in two different rules, such as node-specific and network-specific managements.

The authors in [40] propose SD-WSN6Lo for 6LoWPAN network to enable IoT connectivity. They used Contiki COOJA simulator to follow the SD-WSN paradigm. In this work, the routing between nodes was achieved using control messages such as request message (packet-in), response message (packet-out) and broadcast message (neighbours' message); these messages of control were exchanged between the SDN controller and sensor nodes using the UDP protocol.

2.3. Software-Defined Networking (SDN) Architecture for IoT

In general, the separation of the control plane from the data plane is the essential idea of the SDN concept and how it is distinguished from traditional networks. This process is called the abstraction of the control plane and data plane. In the past, the autonomous system used the traditional network, where each switch or router made decisions based solely on the local logic regarding how to forward the packets. In this case, with SDN, a more centralised model brings direct software programmability to the network. This concept is based on a central controller that can manage and monitor network behaviour. OpenFlow was designed in SDN to allow applications to manage network devices with software that works on servers and communicates directly with switches rather than the control of the switches or routers [19].

Our SDN structure is designed in three planes [12], as shown in Figure 2.1.: the application, control, and data plane. The application plane is the place of the IoT applications need without knowing the basic infrastructure of the network. The data plane is responsible for packet forwarding using the IoT devices based on the IEEE 802.15 protocol. The control plane is the place to control routing decisions for each network device.

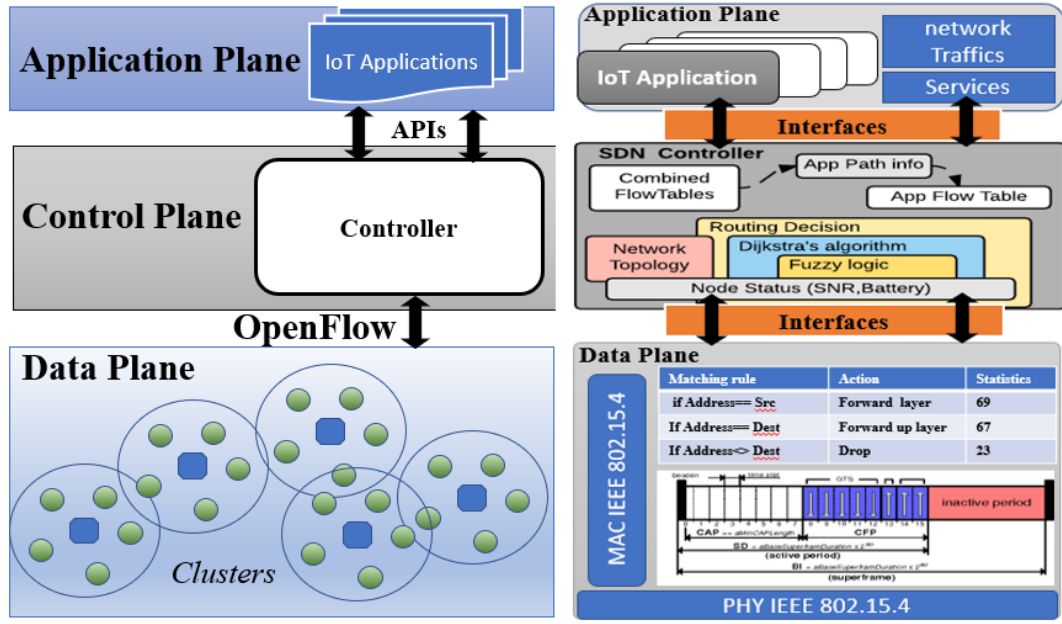


Figure 2.1. The Architecture of the SDN-based WSN clusters for IoT

As shown in Figure 2.1., the proposal framework has three layers, including IEEE 802.15.4 used by IoT sensor nodes to collect data, and SDNC as a flexible network to meet IoT applications' requirements.

2.3.1. SDN application plane

The application plane is the place to meet the needs of IoT applications without knowing the underlying network infrastructure. It includes the requirements and services of the IoT applications [19] [41]. IoT applications rely on the performance of forwarding and routing methods to achieve their services and tasks within the network. Therefore, the SDN controller must make the appropriate decision based on the lowest possible cost from source to destination sensors. To send data from source to target sensors, there must be a specific path between the nodes. For this reason, the application's path should be as short as possible and consider the energy of the sensors.

In our system, the path is built from sensor statuses using smart models in the SDN controller. Moreover, the SDN controller allows applications to configure the network from the application plane instead of the distributed policy that requires individual

configuration. For that, SDN sets up APIs to allow communication between the application plane and control plane. The different applications use the APIs in the SDN stack to formulate new flow rules to the controller.

2.3.2. SDN control plane

The control plane handles configuration management for SDN-conformable devices and realises the network topology. The control plane configures connection paths or flows into the data plane. The control plane connects the application plane and the data plane [40]. It receives the IoT application's tasks from the application plane and converts them into instruction sets, which are sent to the data plane using the OpenFlow protocol [33]. It provides the application plane with a comprehensive view of data layer resources, such as status and attributes. The control plane is the right place to control the routing decision for each device in the network [23].

In our proposed system [12], the SDN controller in the control plane contains many models, such as the network topology, routing decision model, and node status model. The network topology is used to sort the IoT devices' data into clusters and make a routing decision, using the Dijkstra algorithm and fuzzy logic models to perform smart routing based on energy. Node status is used to collect the status of the IoT devices. In addition, the data structures are organised in a combined flow tables model consisting of three classes: network topology, flow table, and nodes classes. The details of these classes are explained in later chapters.

2.3.3. SDN data plane

The data plane authorizes the SDN controller to manage and control the resources of the IoT forwarding devices [23]. Additionally, the data plane is responsible for forwarding packets to the selected target. The data plane in our proposed model consists of the IoT devices using the IEEE 802.15.4 standard protocol; these devices are the main means of data collection, collecting data such as the status of the devices. The SDN controller injects the flow entries into the IoT forwarding devices using

OpenFlow protocol. The data plane devices apply the required actions of the entries to the forwarded incoming based on matching rules. Therefore, IoT forwarding devices forward or drop a packet based on the flow entries from the control plane. The SDN can also update or replace flow entries with new entries when the IoT application requirements change.

2.3.3.1. IEEE 802.15.4

IEEE 802.15.4 is the most important communication technology for low-power wireless networking [42]. In IEEE 802.15.4, the medium access control (MAC) protocol enables the transmission of MAC frames using the physical channel, which supports two operational modes: non-beacon- and beacon-enabled modes. In the non-beacon-enabled mode, which does not support beacon and superframe mechanisms, medium access is controlled by an unslotted CSMA/CA protocol. In the beacon-enabled mode, which supports beacon and superframe mechanisms, the coordinator or router periodically sends beacons to synchronise nodes associated with its own cluster coverage area. However, this mode also enables the allocation of contention free period (CFP) slots, called guaranteed time slots (GTSs) for nodes that provide a timing guarantee for bandwidth.

2.3.3.2. Superframe Structure

The superframe impacts upon the performance and quality of services on the network [43]. The superframe structure includes an active and an inactive period, determined for one cycle between two beacon frames. The superframe structure is presented in Figure 2.2.; the beacon interval (BI) defines the time between two consecutive beacon frames and contains an active and inactive period. In the superframe active period, the beacon frame starts the transmission at slot 0. The superframe duration (SD) specifies the active portion of the BI period; the length of the active period is determined manually by the SD settings from the upper layer. Superframe duration is divided into 16 equally sized time slots, during which frame transmissions are allowed. Superframe duration contains the contention access period (CAP), which is used to share the

channel between nodes competitively and the CFP that used to allocate GTS for some nodes. The CFP starts immediately after the CAP has finished and must be completed at the end of the active period.

$$\left. \begin{aligned} BI &= aBaseSuperframeDuration \times 2^{BO} \\ SD &= aBaseSuperframeDuration \times 2^{SO} \end{aligned} \right\} \text{for } 0 \leq SO \leq BO \leq 14 \quad (2.1)$$

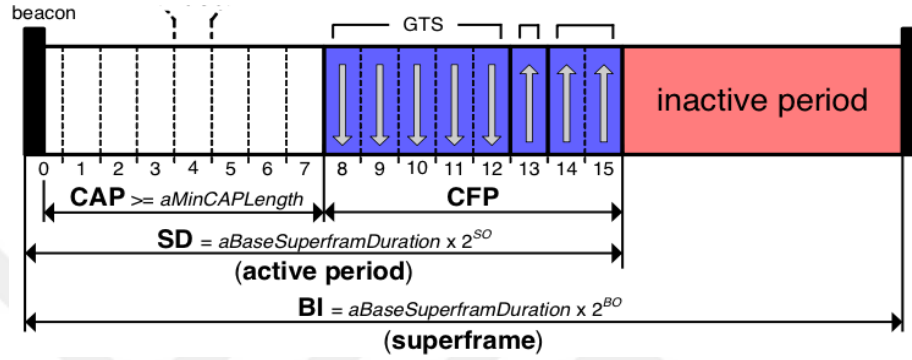


Figure 2.2. IEEE 802.15.4 superframe [43]

Beacon order (BO) and superframe order (SO) are used to change the length of BI and SD. These parameters must be previously allocated by coordinator or router [44]. The current WSN configures these parameters manually, which does not fit the changes of network size, neighbours, and number of nodes.

In our proposed system, these configurations are at the control plane. During topology discovery, the SDN controller (SDNC) assigns BO and SO parameters according to current network size, neighbours, and number of nodes, which are discussed in more detail in subsequent chapters. Table 2.1 shows the effect of BO and SO values on beacon interval and superframe duration. This period defines the start time of the child node in order to start sending data or command frame.

Table 2.1. Beacon interval and superframe duration evaluation

BO/SO	Equation 2.1	BI/SD start time
1	$960 \times 2^1 \times 4 / 250000 =$	0.03072
2	$960 \times 2^2 \times 4 / 250000 =$	0.06144
3	$960 \times 2^3 \times 4 / 250000 =$	0.12288
4	$960 \times 2^4 \times 4 / 250000 =$	0.24576

In the inactive period (sleep time) of the superframe, the child node starts a transmission according to its start time. For example, when SDNC BO=4 SO=1 is applied to (Equation 2.1), the SDNC starts at 0 time and sleep time will be available to several child nodes whose BO=SO=1, as shown in Figure 2.3.

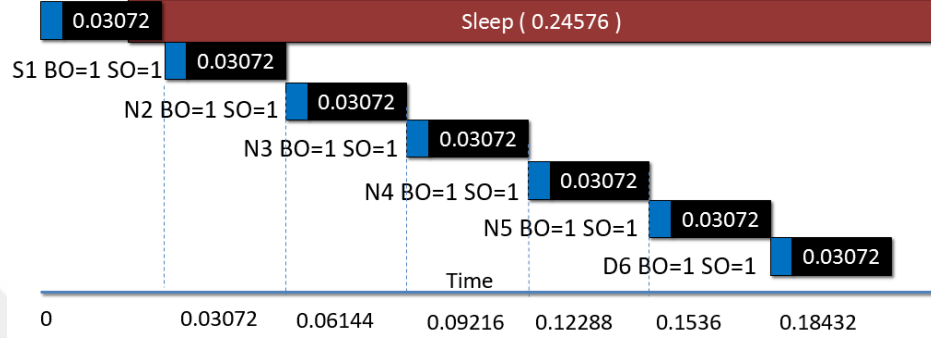


Figure 2.3. Superframe inactive period (sleep time)

2.3.4. OpenFlow (WSANFlow) protocol stack

OpenFlow is a protocol that relies on flow tables, developed by researchers into SDN [45][46]. Since OpenFlow's origins, it has been based on the idea of creating forwarding tables to guide packets from source to target through the next hop, and determines the actions to be taken on the flow [26]. Our WSANFlow protocol uses the forwarded tables between the control plane and the data plane. On the data plane, the flow table contains flow entries that are redirected between the forwarding IoT devices. Each flow entry includes match rule field, actions, and statistics; according to these entries, it will decide how to handle incoming packets.

CHAPTER 3. THE SDNC FUZZY-BASED ROUTING DECISION

3.1. Introduction

In this study, we propose a new fuzzy-based routing discovery algorithm inside the SDNC for clustered networks over an IEEE 802.15.4 superframe structure, supporting many applications' requirements. Fuzzy logic can play an important role in allowing SDNC to handle routing decisions intelligently. Using fuzzy logic to improve network connection costs by considering available network edges as neighbour devices' signal-to-noise ratio (SNR) and devices' energy levels collected from device statuses. This is intended to integrate these elements into a uniform cost for later use in the Dijkstra algorithm to create an IoT application path. Each application has a source and a destination sensor [47][48]. The source device represents the detected sensor device that needs to send data to the destination. The destination device represents the sink or gateway device to forward data to internet access.

The main objective of this study is to establish the path with the minimum energy cost resulting from fuzzy-based Dijkstra's algorithm, the other devices not in this path can keep their energy using sleep time from the schedule mechanism cluster-tree WSN [49]. The limitation on the battery power lifetime is a challenging issue for sensor networks, due to the very high cost of battery replacement and the fact that some sensors cannot be replaced. Therefore, efficient routing protocols over WSN must counter important issues such as unnecessary power consumption and absence of fault tolerance.

This study also presents a new method for routing decisions in SDWSANs: during the transmission time, an interruption according to any device's energy level will inform the SDNC to change the existing application's path. Once the best path has been built

up with the smallest energy cost, other devices keep their battery energy, which can be used by another application.

3.2. The SDNC Fuzzy-based Routing Decision

Routing on the cluster-tree network topology is a new challenge for forwarding packets between EDs to reach the ED destination. Because the network operates in the cluster-tree structure and superframe structure, the cluster can have active mode or sleep mode during runtime. Therefore, the forwarding data process must work between the active clusters; from one active cluster to another active cluster to avoid dropping packets during the cluster sleep mode.

The study proposes a new approach which adds fuzzy logic into routing to improve SDNC decisions regarding cluster-tree topology. In SDNC, Dijkstra's algorithm uses the links cost resulting from fuzzy logic to generate the shortest path from the application's source requests to the destination (gateway), with associated energy in this path according to the fuzzy rule base in Table 3.1. These paths distribute energy between devices using flow tables which take into consideration the number of active clusters and the application's ID. Moreover, in SDNC, node status aggregation (NSA) receives ED statuses; statuses such as device energy level and neighbour device SNR are updated periodically in real time.

In network topology, devices broadcast beacon frames; each device has a beacon frame and reacts with its own beacon frame to SDNC in order to update NSA. Moreover, the devices add to the beacon frames their devices' battery levels and SNR, and the addresses of neighbour devices.

In fuzzy-based Dijkstra's, if there is a change in devices' energy levels past a certain threshold, SDNC will be interrupted to reselect another device by redefining the application path, in order to gain energy across the total network topology. In regular Dijkstra's, the path does not change, and the path selected will remain fixed over time once selected. During transmission time, the intermediate devices in this path consume

much more energy, and other neighbour devices in the topology still have a high level of energy, meaning the path must be reconfigured and reselected, incurring new costs. Reselecting paths will make a lifetime balance across the entire network; devices that expend a great deal of energy will enter sleep time to save energy. The SDNC has the ability to reconfigure WSANs after deployment using open flow protocol.

In addition, fuzzy logic and Dijkstra's are working together in order to have a greater effect on energy for the routing process. Dijkstra's algorithm calculates the link cost output based on fuzzy logic. The fuzzy system includes two inputs and one output (more details in Section 3.6.). The simple procedures are summarised in Figure 3.1. on the cluster-tree architecture.

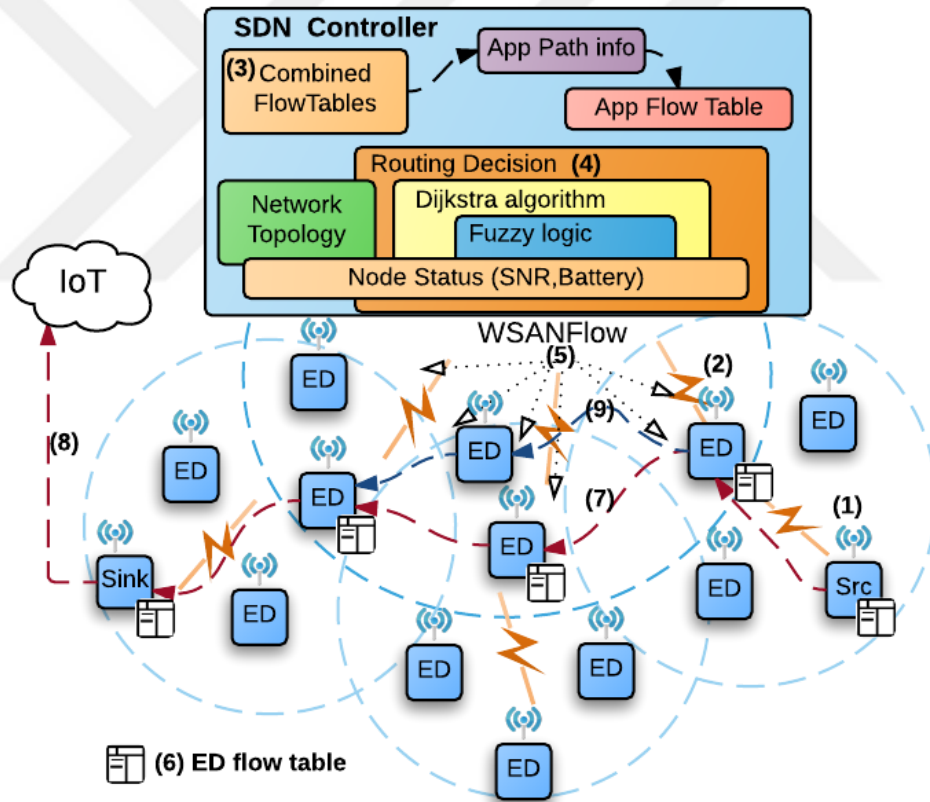


Figure 3.1. Architecture design of routing protocol in the SDN controller

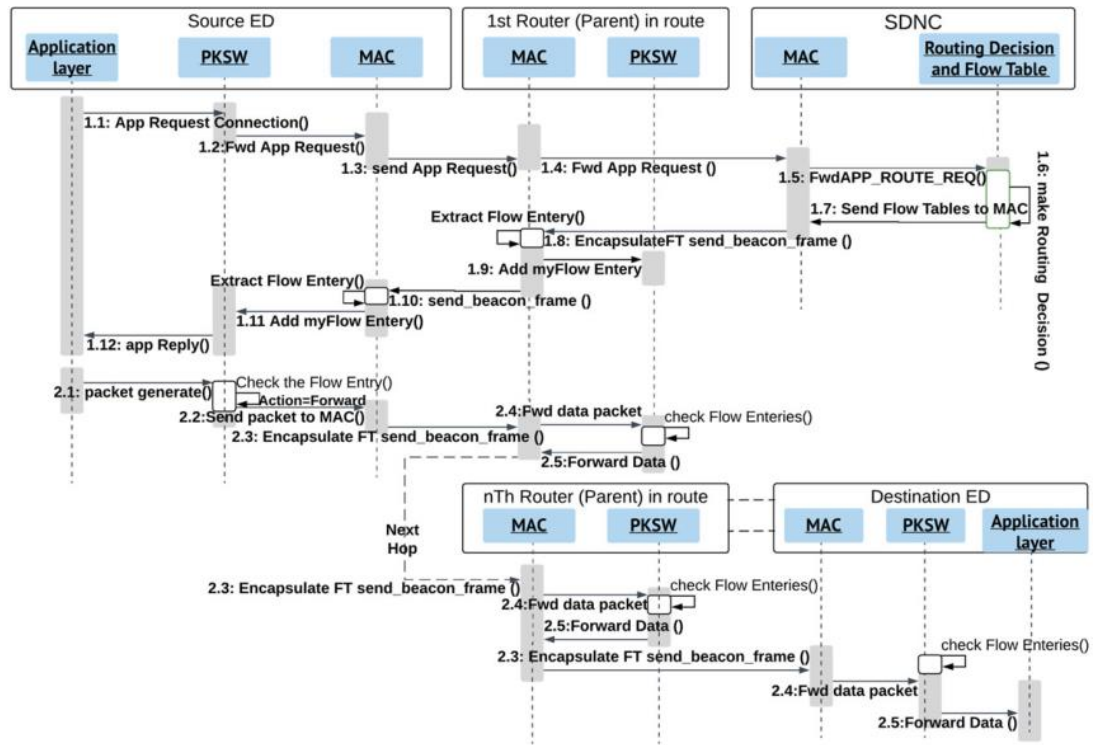


Figure 3.2. Sequence diagram of the system processes

When an application wants to send data over the network, the following steps are generally performed, as shown in Figure 3.1. and Figure 3.2.:

- 1) Application uses SRC-ED to send a request connection to SDNC towards parent ED address.
- 2) Parent ED forwards request to SDNC via intermediate ED addresses.
- 3) SDNC records the application info and connection requests from the ED sources and uses EDSA to read network topology info. The intelligent network module receives this request and checks the available network resources. At this moment, the SDNC should be fully aware of the network and its EDS states, so that it uses certain algorithms to arrive at the most appropriate decision for a specific route for the current transmission request.
- 4) SDNC collects SNR and battery values by NSA (see Section 3.5.) to make a routing decision based on link costs using fuzzy logic (see Section 3.6.) and performs routing using Dijkstra's algorithm (see Section 3.7.) to generate the shortest path.

- 5) SDNC converts the resulting path to WSAFlow [4] in order to distribute flow entries to the related EDs.
- 6) Each ED checks the flow entry and adds its own entry.
- 7) Source ED begins to send the data packet through its flow entry to reach the sink ED (gateway).
- 8) Sink sends data as a gateway to IoT cloud.
- 9) An alternative path is reselected by SDNC routing decision during running time.

This chapter, the most important in this thesis, describes the components of the SDN controller. The SDN controller guides and manages the network in a smart way to make the right decisions. As can be seen in the figure, the SDN controller contains five main components: combined flow tables, network topology, node status, Dijkstra's algorithm, and fuzzy logic.

3.3. Combined Flow Tables

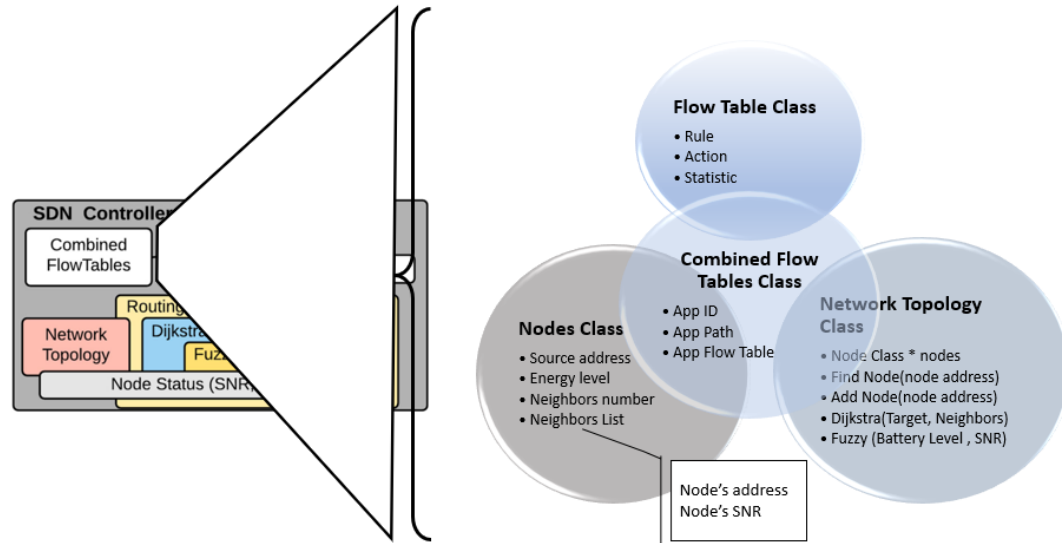


Figure 3.3. Combined flow table class

The combined flow tables component consists of three main classes, structured as a C++ object-oriented program, as shown in Figure 3.3. The node class includes all the codes and functions related to nodes such as address, energy level, number of neighbours, and their data. Flow table class includes data structure and code about the

flow tables and entries, whose objects are created from this type to other classes. The network topology class includes the functions of node operations on the network, Dijkstra's algorithm, and fuzzy logic algorithm, about which we will see more details in the following sections. Objects are created from different classes according to the needs of the functions and procedures required to perform the necessary tasks. When an application request is sent to the SDNC for data transmission, SDNC uses the combined flow tables class to create application's path and to link this to flow tables. Each application can have a special path in the network from source to target sensors. The path is built depending on the node, flow table and network topology classes, generated by the smart algorithms in the SDN controller.

3.4. Network Topology

Network topology uses the topology discovery method to determine the clusters in the network which collect the status information of all nodes and their neighbour list information. In the network discovery process, when nodes see the beacon frame, they respond with their own beacon frame to join the network. The responding node adds into the frame the node battery level and SNR as a distance from other neighbour nodes, including their addresses.

The network topology sets up the SDN controller beacon settings, such as BO and SO parameters, and starts time. These settings determine the sleep time of each cluster according to its number of child nodes. Meanwhile, the type of nodes (forwarding node or child node) in the network is determined by SO and BO parameters. For example, if the node has $BO=SO=1$ or more, then it is a forwarding node; if the node has $BO=SO=-1$, then it is a child node. As shown in Figure 3.4., the network topology performs this process by checking all nodes so that the settings are given only to the nodes with children in their clusters. Eventually, these settings determine the length of the active superframe duration SD to fit the number of forwarding devices in the network.

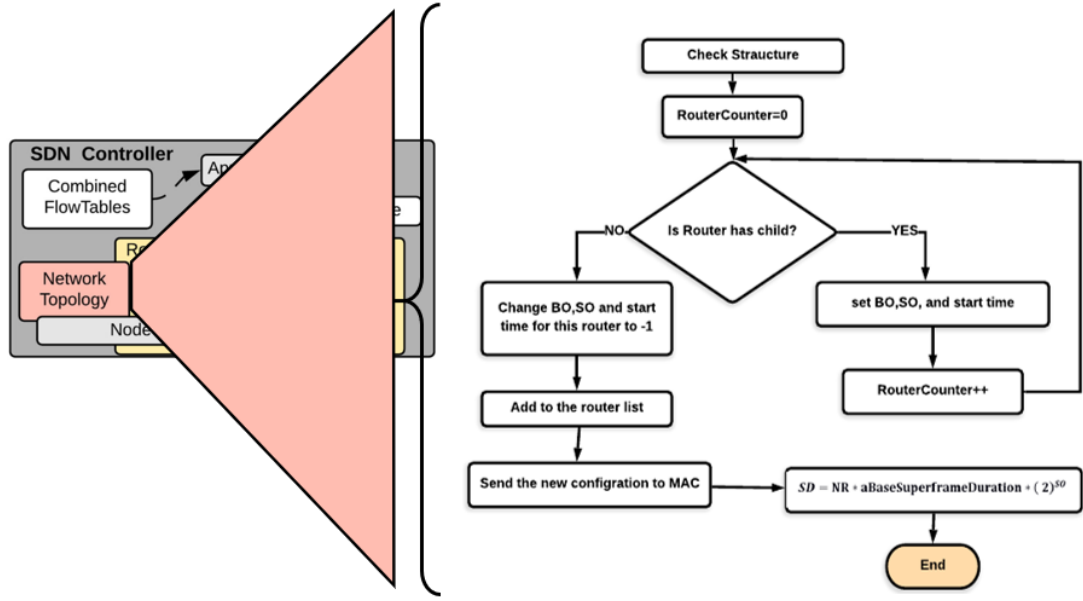


Figure 3.4. The network topology

3.5. Node Status Aggregation (NSA)

When the network starts to work, the broadcasting packets process starts between all devices, according to the settings determined by the network topology. At the first time, the SDNC network discovery dynamically generates a time slot (i.e. BO and SO) for each cluster device with children devices, which makes devices broadcasting beacon frames. Each device receives a beacon frame and reacts with its own beacon frame to SDNC in order to update node status aggregation (NSA). Moreover, the EDs insert their device's battery level and SNR and the addresses of neighbour devices into the beacon frames. In this case, the beacon frames are distinguished from other beacons using NDP flag field. Once a device receives the neighbour device beacon frame, it generates a command frame and adds device status information, then sends it to SDNC. Moreover, each device stores a neighbour list containing a list of all SNR values for each connection between the device and its neighbourhood addresses, resulting from network discovery. Subsequently, when the SDNC receives all devices' command frames, NSA begins to work, as shown in Figure 3.5.

In Figure 3.5., as each node status arrives at the node status algorithm, it checks the arrived node status against the node class list. Moreover, each node stores a neighbour

list containing a list of all SNR values for each connection between the node and its neighbourhood addresses. In order to keep the neighbour list up to date, the SDNC node status algorithm compares the incoming node status to its node class list and adds it if a node does not exist. Otherwise, it will update the node status information in the list. The battery level is overwritten each time a packet is received, because each node consumes battery and resends the packet. At that time, if the ED battery level is less than the threshold, it will interrupt to reroute the path (making a routing decision) by SDNC using fuzzy-based Dijkstra's algorithms and generate a new path including EDs with higher battery levels. Subsequently, a list node containing all the nodes' statuses is sent to the Dijkstra fuzzy algorithm to create a topology cost based on this list.

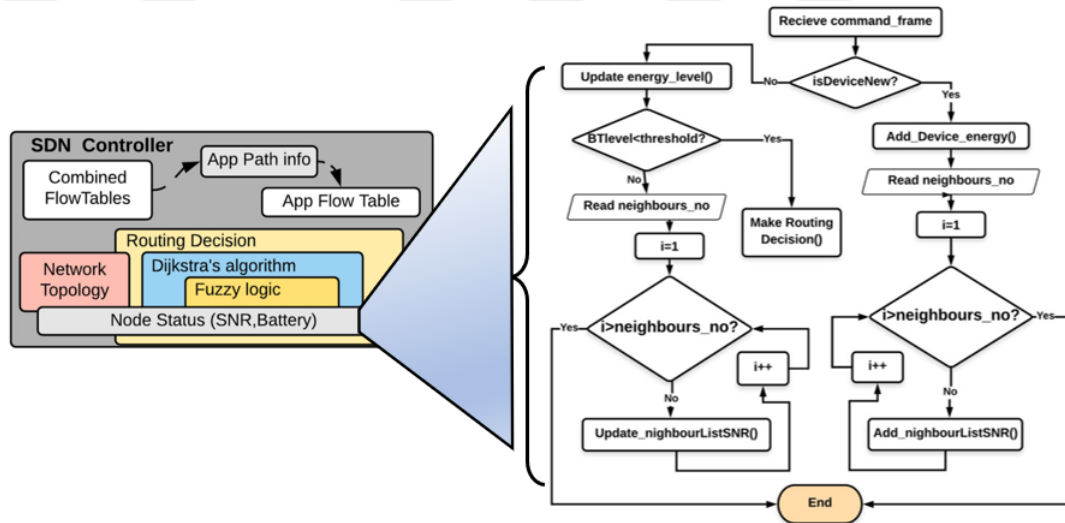


Figure 3.5. Node status aggregation

3.6. Fuzzy Logic

Fuzzy logic [50] is an active system of artificial intelligence which plays an important role in the SDNC to extend the network lifetime; fuzzy logic with inputs and output is illustrated in Figure 3.6. Fuzzy logic is used to improve network connection costs by considering the available network edges (link quality, power consumption, link cost, etc.).

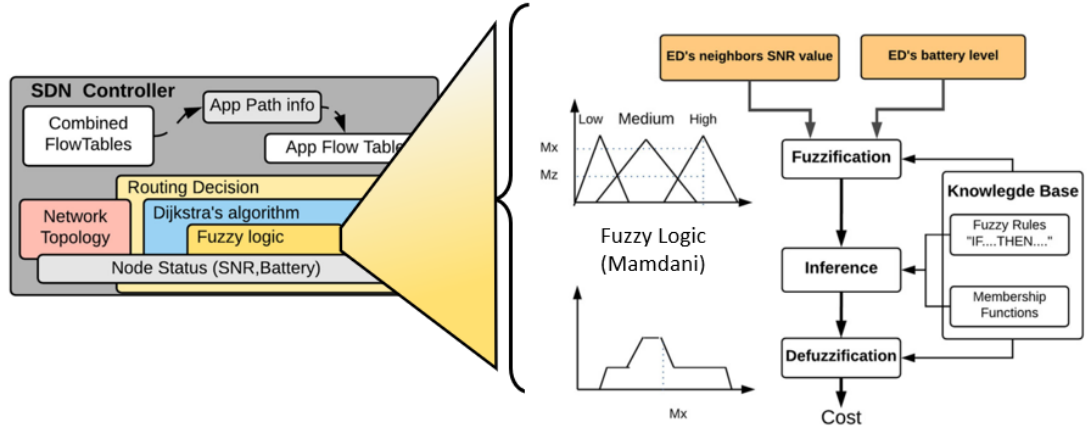


Figure 3.6. Fuzzy logic with inputs and output

3.6.1. Fuzzification

Fuzzification checks the input values of SNR and battery level into input fuzzy set values; each fuzzy set has several membership functions denoted by linguistic variables. The SNR fuzzy set has three membership functions (weak, medium, and strong) and the battery fuzzy set has three membership functions (low, medium, and high), represented in Figure 3.7. as the input membership functions.

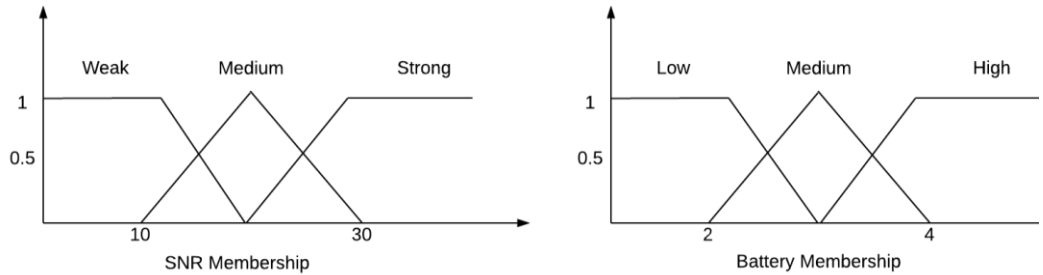


Figure 3.7. Fuzzy inputs SNR and battery level memberships

When fuzzy logic is activated, the input values are calculated for fuzzy input sets of the membership functions (Equations 3.1. and 3.2.). An example of determining the relevant fuzzy sets is shown for fuzzy input data (SNR, battery level) = (16, 4). In Figure 3.8., the example of SNR value belongs to the weak and medium membership functions and the battery level value belongs to the high membership function:

$$SNR(x1) = \sum_{i \in MSNR}^3 \mu_i(x1)/i \quad (3.1)$$

Using (Equation 3.1) $SNR(16) = 16/\text{Weak} + 16/\text{Medium} + 0/\text{Strong}$

$$\text{Battery}(x2) = \sum_{i \in MBt}^3 \mu_i(x2)/i \quad (3.2)$$

Using (Equation 3.2) $\text{Battery}(4) = 0/\text{Low} + 0/\text{Medium} + 4/\text{High}$

Where i is the linguistic (low or weak, medium, strong or high).

```

SNR: Value= 16
  L: Value (20)      [ 0 -> 20]
  M: Value (30)      [ 10 -> 40]
  H: Value (0)       [ 20 -> 50]

Battery: Value= 4
  L: Value (0)       [ 0 -> 3]
  M: Value (0)       [ 2 -> 4]
  H: Value (50)      [ 3 -> 5]

```

Figure 3.8. SNR, battery membership functions map the input values

In fuzzification, the activation process for antecedents' SNR and battery level is computed by the membership degree function. Fuzzy operation can be calculated using the membership compute degree ($Mf(v)$) function, as given by (Equation 3.3). We can apply it in this example as follows:

$$\mu_f(x) = \min \left(\min \left(\maxValue \times \left(\frac{x-a}{b-a} \right), \maxValue \times \left(\frac{d-x}{d-c} \right) \right), \maxValue \right) \quad (3.3)$$

Using (3.3), we can explain the examples of SNR and battery level values and apply this to the resulting membership degree functions. For SNR, a weak membership function, which has a range of $\{a, b, c, d\} = \{0, 10, 10, 20\}$, the $SNR_{weak}(16)$ is represented by

$$\begin{aligned} \mu_f(16) &= \min \left(\min \left(50 \times \left(\frac{16-0}{10-0} \right), 50 \times \left(\frac{20-16}{20-10} \right) \right), 50 \right) \\ \mu_f(16) &= \min (\min(50 \times 1.6, 50 \times 0.4), 50) \\ \mu_f(16) &= \min (\min(80, 20), 50) \\ \mu_f(16) &= \min(20, 50) \end{aligned}$$

$$\mu f(16) = 20$$

For an SNR medium membership function, which has range of $\{a, b, c, d\} = \{10, 20, 20, 40\}$, the $SNR_M(16)$ is represented by

$$\mu f(16) = \min(\min\left(50 \times \left(\frac{16 - 10}{20 - 10}\right), 50 \times \left(\frac{40 - 16}{40 - 20}\right)\right), 50)$$

$$\mu f(16) = \min(\min(50 \times 0.6, 50 \times 1.2), 50)$$

$$\mu f(16) = \min(\min(30, 56), 50)$$

$$\mu f(16) = \min(30, 60)$$

$$\mu f(16) = 30$$

For a battery high membership function, which has a range of $\{a, b, c, d\} = \{3, 4, 4, 5\}$, the $Battery_H(4)$ is represented by

$$\mu f(4) = \min(\min\left(50 \times \left(\frac{4 - 3}{4 - 3}\right), 50 \times \left(\frac{5 - 4}{5 - 4}\right)\right), 50)$$

$$\mu f(4) = \min(\min(50 \times 1, 50 \times 1), 50)$$

$$\mu f(4) = \min(\min(50, 50), 50)$$

$$\mu f(4) = \min(50, 50)$$

$$\mu f(4) = (50)$$

The results of the SNR weak, SNR medium and battery high membership function for this example is shown in Figure 3.8. Input variables are assigned degrees of membership in various classes. The purpose of fuzzification is to map the inputs from a set of sensors to values from 0 to 1 using a set of input membership functions.

3.6.2. Rule evaluation

A fuzzy inference produces a relationship between input and output fuzzy sets using a knowledge base that contains fuzzy rules and membership functions. The rule consists of two main parts: the ‘If’ side and the ‘Then’ side: If (antecedent) Then (consequent). Table 3.1. shows nine rules of the fuzzy rule to produce an output fuzzy set linguistic value. These rules are structured as a condition (IF-THEN).

Table 3.1. Fuzzy logic rule base

Rule Base	IF-side (Antecedents)	THEN-side (Consequence)
1	(SNR is weak) and (Battery is low)	(Cost is VeryHigh)
2	(SNR is Weak) and (Battery is Medium)	(Cost is Average)
3	(SNR is Weak) and (Battery is High)	(Cost is Low)
4	(SNR is Medium) and (Battery is Low)	(Cost is High)
5	(SNR is Medium) and (Battery is Medium)	(Cost is Average)
6	(SNR is Medium) and (Battery is High)	(Cost is Low)
7	(SNR is strong) and (Battery is Low)	(Cost is High)
8	(SNR is strong) and (Battery is Medium)	(Cost is Medium)
9	(SNR is strong) and (Battery is High)	(Cost is VeryLow)

As mentioned in our previous example, the SNR is between weak and medium and the battery is high, which means that the 3 and 6 rules from Table 3.1. can be applied; in this case, the rule evaluation is shown in Figure 3.9. Before the rules can be evaluated, the inputs must be fuzzified according to each of these linguistic sets.

Rule #	: If [SNR]	and [Battery]	Then [Cost]
Rule #1:	[20]	[0]	[0]
Rule #2:	[20]	[0]	[0]
Rule #3:	[20]	[50]	[30]
Rule #4:	[30]	[0]	[0]
Rule #5:	[30]	[0]	[0]
Rule #6:	[30]	[50]	[30]
Rule #7:	[0]	[0]	[0]
Rule #8:	[0]	[0]	[0]
Rule #9:	[0]	[50]	[0]

Figure 3.9. Rule evaluation results

Rule evaluation is part of the implication method which defines the rule's conclusion. We use the implication method by taking the minimum degree of membership for two sets as $\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$. The activation degree process in the implication method takes the minimum degree of the variables among the SNR and battery memberships involved in the rule evaluation as the cost membership degree. We use the activation degree to multiply them with the linguistic in terms of consequences to determine the rule's conclusion [51].

Consider the example in Figure 3.9.: according to the rule match found in Rule 6, the activation degree trims the SNR (weak and medium memberships) and battery (high membership) by taking the minimum value between SNR and battery, [cost = min (30, 50)], as the cost (low membership). Hereafter, in order to find the output value for each rule involved, the aggregation process sums up the membership functions into the K variable by taking the maximum of the membership functions of the resulting cost. The K value will be used later in the defuzzification process. Therefore,

$$K=30= \max (\min (\text{SNR}, \text{Battery}), \text{cost}) \quad (3.4)$$

3.6.3. Defuzzification

The last step is known as the defuzzification process, in which the fuzzy output is converted to a single value. Figure 3.10. shows the fuzzy output of a cost set with six membership functions (very low, low, medium, average, high, and very high).

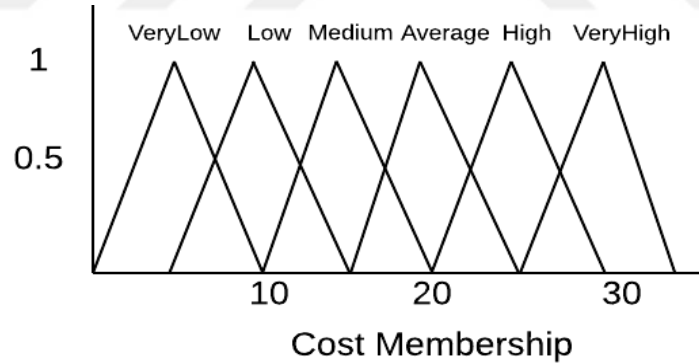


Figure 3.10. Fuzzy output cost membership

The Mamdani system is used in the defuzzification process as a typical module, and the centre of gravity (CoG) method is used to resolve conflict issues such as ‘cost is medium/low’, using membership functions. The CoG method computes the centre of the area below the curve to find a final output for the cost’s crisp, as described in (Equation 3.5).

$$cost_{crisp} = \frac{\sum_{i \in Cost_{\mu f}}^6 (Area\ i * Centroid\ i)}{\sum_{i \in Cost_{\mu f}}^6 Area\ i} \quad (3.5)$$

Where $i = \{1, 2... 6\}$ is the output membership, and $Area$ is computed for each output membership. $cost_{crisp}$ is the final desired output. As a result of the previous example, we substitute the $K=30$ value in (Equation 3.4) and the low cost membership resulting from rules 3 and 6. The following equations convert the fuzzy output to the cost crisp value as final results:

$$base1 = d - a \quad (3.6)$$

$$base2 = base1 - \left(\frac{K}{Lv1}\right) - \left(\frac{K}{Lv2}\right) \quad (3.7)$$

$$Area = K * \frac{base1 + base2}{2} \quad (3.8)$$

$$Centroid = a + \frac{d-a}{2} \quad (3.9)$$

Figure 3.11. shows the implementation of the previous example on these equations. The range of $Cost_{Low}$ membership is $Cost_{Low} \{a, b, c, d\}$, represented in our example as $\{5, 10, 10, 15\}$, where the first and last values, a and d , are used as inputs in the (Equations 3.6, 3.7, 3.8 and 3.9).

```
=====
K =30
a =5
d =15
base1=10
LinguisticVlaue1=10
LinguisticVlaue2=10
base2 =4
Sum of Area X Centroid = 2100
Sum of Area = 210
Cost -> crisp = Sum of Area X Centroid / Sum of Area = 10
=====
```

Figure 3.11. Example fuzzy results

The flowchart of the defuzzification procedures is shown in Figure 3.12.

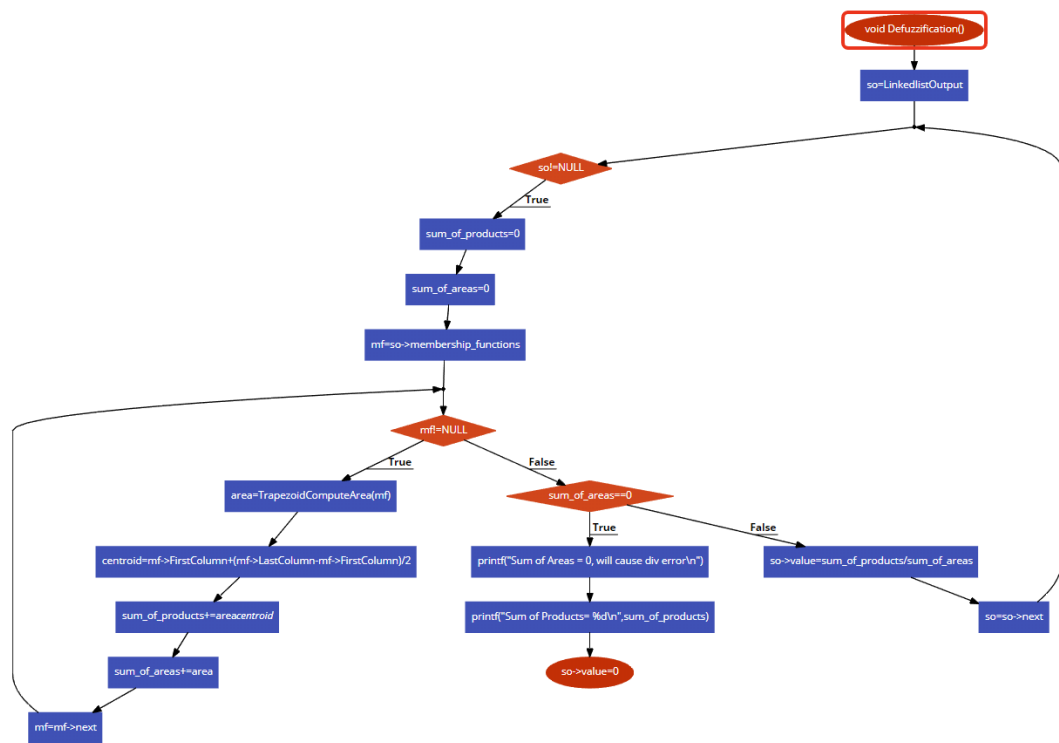


Figure 3.12. The flowchart of the defuzzification procedures

Now let us see how the battery level changed for this ED and the effect that accompanies cost value during the network life cycle. Through the process of transmitting and receiving, the device gradually loses its energy, as shown in Table 3.2.:

Table 3.2. Cost vs battery level

Cost	SNR	Battery level
10	16	5
10	16	4
20	16	3
27	16	2
27	16	1

As seen in the table, the low cost is the best candidate for Dijkstra's algorithm for building application paths, therefore the ED is selected based on the cost value of the connection link between a device and its neighbour. However, when the level of energy is lower, the output cost value is higher, which means that it is not a good

candidate for Dijkstra's algorithm, because as we will see later, Dijkstra's algorithm prefers to select the minimum values of these connection links between ED and its neighbours. In this way, we have established the relationship between cost and energy level in fuzzy rules, as illustrated in Table 3.2.

3.7. Dijkstra's Algorithm

Dijkstra's algorithm [52] is an algorithm which finds the route with the lowest possible cost in order to determine the best path from source to destination node in the network. When the SDNC receives the status of each node, the NSA in SDNC starts to collect these statuses for each node, including its neighbours' information in the class node representing the collected information of all nodes. if there is a connection request from an application or an interrupt from NSA to change the route according to the ED battery level, Dijkstra starts working with the fuzzy logic shown in Figure 3.13. Dijkstra's input is the cost for each connection resulting from fuzzy logic. A lower cost value means that the ED neighbourhood is very close to the device, while a high cost value means that the ED neighbourhood is far away; these decisions are determined by the fuzzy rule base in Table 3.1.

However, SDNC is able to dynamically build the network topology and select the best route between nodes based on the resulting cost from the fuzzy system. As our fuzzy system includes two inputs and one output, when these fuzzy input values change then cost will change for each node connection. The fuzzy logic inputs are updated when new nodes' statuses arrive at SDNC, then the output cost is inserted concurrently as the input for Dijkstra's algorithm.

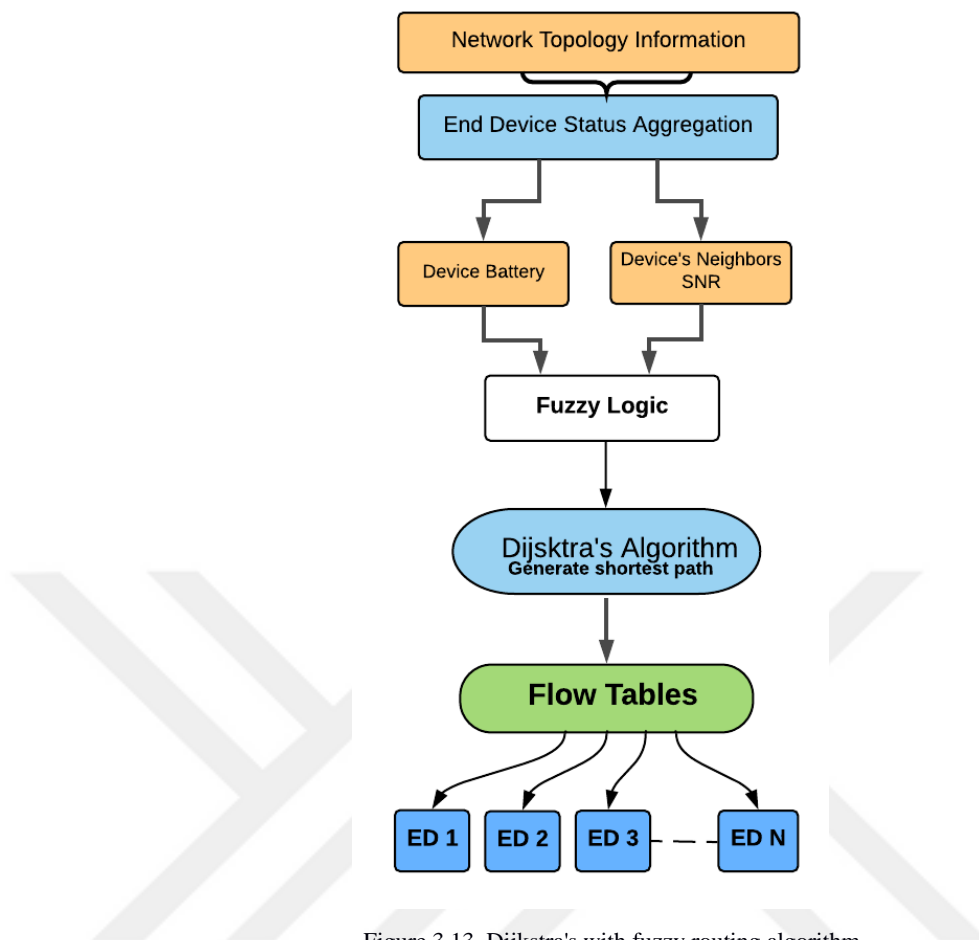


Figure 3.13. Dijkstra's with fuzzy routing algorithm

Dijkstra's algorithm chooses the nearest device to the source using the find device function and runs over neighbours in the loop. Subsequently, it compares the nearest device with its neighbours, selected from the devices list presented in Figure 3.14. (Steps 6, 7, 8, and 9). For each cycle in the loop, the minimum cost for the best path is aggregated as the total path cost from the EDs to source ED which Dijkstra's algorithm returns.

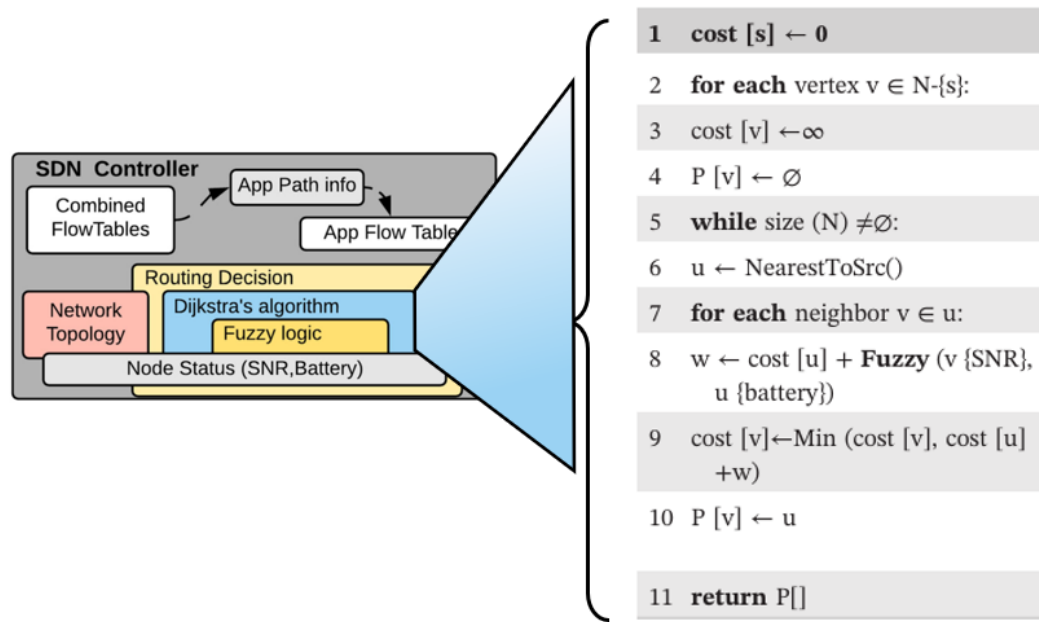


Figure 3.14. Dijkstra's algorithm

3.7.1. Flow tables layout

Table 3.3. shows a flow table formed by the SDNC; the entries represent each application that resulted in its path from Dijkstra's algorithm. Each ED address in the application's path receives an entry contained next hop and an action to forward or drop packets arrived.

Table 3.3. Flow table entries

Application ID	Device Address	Matching Rule		Action		Statistics	
		Comparator	Src Address Match Rule	Forward / Drop	Next Hop	Cluster Dir	Number of Packets
1	13	'=	13	Forward	2	0	34
1	4	'≠	1	Drop	-	-	81
2	5	'=	1	Forward	4	0	33
2	4	'≠	1	Drop	-	-	94
3	6	'=	7	Forward	4	0	32
3	15	'≠	7	Drop	-	-	81
2	5	'=	2	Modify	8	0	65

Scenario 1

```
nearest=12cost before for node=4cost=24
cost After for node=4cost=24
Cost from the Source: 32
12 4 2 1 11
```

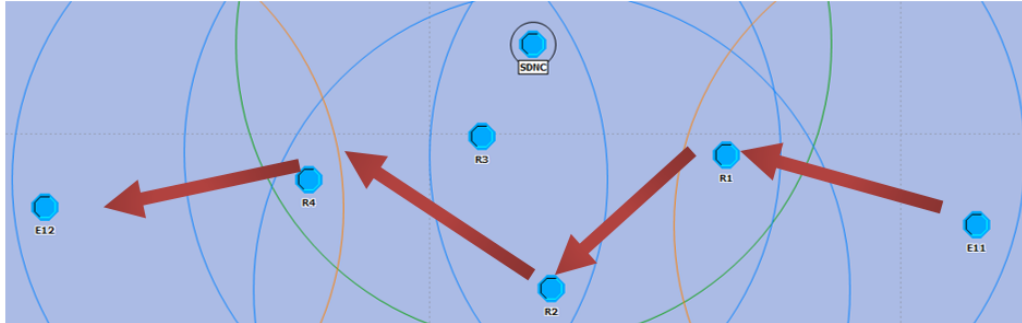


Figure 3.15. Network topology Scenario 1

As seen in Figure 3.15., the first network topology scenario represents SDNC and six nodes. In this scenario, the IoT application wants to send data from the source ED with address 11 to destination ED with address 12, and the Dijkstra's algorithm result of our system as total end-to-end costs is 32, with a path running from 11=>1=>2=>4=>12.

Scenario 2 and R3moved

```
cost After for node=12cost=32
nearest=12cost before for node=4cost=24
cost After for node=4cost=24
Cost from the Source: 32
12 4 3 1 11
```

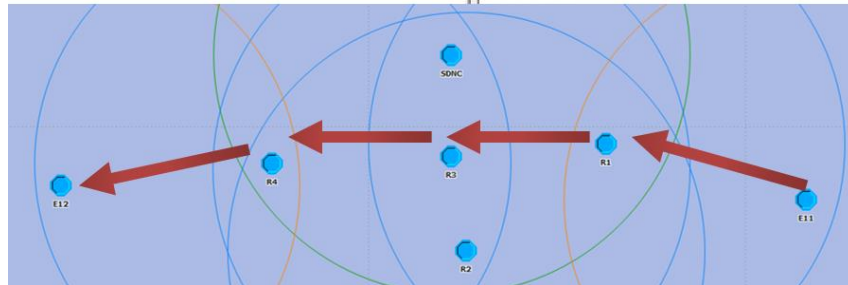


Figure 3.16. Network topology Scenario 2

Figure 3.16. shows the second network topology scenario, in which we change the position of node R3 in order to see the change of SNR represented by the distance between nodes. The Dijkstra's algorithm result of this scenario shows that the path has changed to 11=>1=>3=>4=>12 and the total end-to-end cost remains the same because of the vertically equal distance and energy to the first scenario.

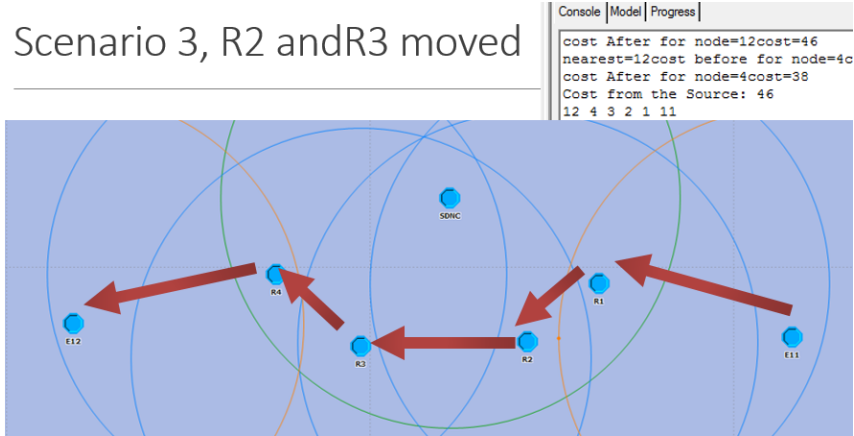


Figure 3.17. Network topology Scenario 3

Figure 3.17. shows the third network topology scenario, which has position changes to nodes R2 and R3 respectively. The Dijkstra's algorithm result of this scenario as total end-to-end costs is 46 with the path from 11=> 1=>2=>3=>4=>12. In this case, the cost is high because the nodes have limited coverage area and the nodes are far away from each other, so the path becomes longer than before.

3.8. Simulation Results and Discussion

In all experiments, we used a free space channel propagation model in the simulation environment, allowing us to predict the received signal strength of a path with direct line of sight between the transmitter and the receiver. Experiments were performed using fuzzy-based Dijkstra's algorithm, and an SDNC with a regular Dijkstra's algorithm in the first scenario. The second scenario includes a ZigBee-based WSN equivalent. Through these scenarios, simulation results can be extracted to see the effect of the proposed routing discovery algorithm.

3.8.1. Simulation environment: parameters and scenarios

The configurations of the applications' requests to SDNC are shown in Table 3.4., when an ED's battery level passes the threshold, the SDNC alerts an administrator to change the battery of this ED. As shown in Table 3.4., there are four applications, which differ in the start time and packet inter-arrival time for performance testing.

Table 3.4. Simulation parameters

Items	Name	Value
Network Topology	Number of End Devices	10, 20, 30, 40, 50
	Network coverage area	300 m X 300 m
	SDNC location (x, y)	(150, 150)
	Simulation time	10800 s
	Max. No of EDs in Cluster	15
	Path re-establishment threshold	2 J
Device settings for both the proposal and ZigBee-based WSN	Data Rate	250 Kbps
	ED status transmission period	25 s
	Initial Energy	5 J
	Channel model	Free-space propagation model (LoS)
	Power threshold	-76 dBm (80 mW)
Battery parameters (Micaz mode) for both the proposal and ZigBee-based WSN	Transmission Mode (0 dBm)	17.4 mA
	Receive Mode	27.7 mA
	Idle Mode	35 μ A
	Sleep Mode	16 μ A
Application 1 (Src. 11 in Figure 23)	Start time	50 s
	Packet payload size	30 bytes
	Packet inter arrival time	2 s ^x
Application 2 (R 34 in Figure 23)	Start time	200 s
	Packet payload size	30 bytes
	Packet inter arrival time	3 s ^x
Application 3 (R 43 in Figure 23)	Start time	300 sec
	Packet payload size	30 bytes
	Packet inter arrival time	5 s ^x
Application 4 (R 36 in Figure 23)	Start time	400 s
	Packet payload size	30 bytes
	Packet inter arrival time	4 s ^x
ZigBee-based WSN	ZC beacon configurations	BO = 4, SO = 1
	Tree routing configurations	Lm = 2, Rm = 4, Cm = 4
	Number of end devices	50
	Network coverage area	300 m \times 300 m
	ZC location (x, y)	(150, 150)
	Simulation time	10 800 s
	Initial energy	5 J
Application 1 (Src. 2)	Start time	50 s

Abbreviations: ED, end device; SDNC, Software-Defined Networking Controller; WSN, wireless sensor and actuator network.

^x Generated using the exponential distribution function $\exp(\text{mean})$.

3.8.2. Simulation results and discussion

Figure 3.18. shows that the first ED dies after 5500 seconds of simulation time and four applications which are shared among 40 EDs. For one application, the network lifetime is longer than with two or more applications running, because EDs forward data from one source to one destination and traffic is light.

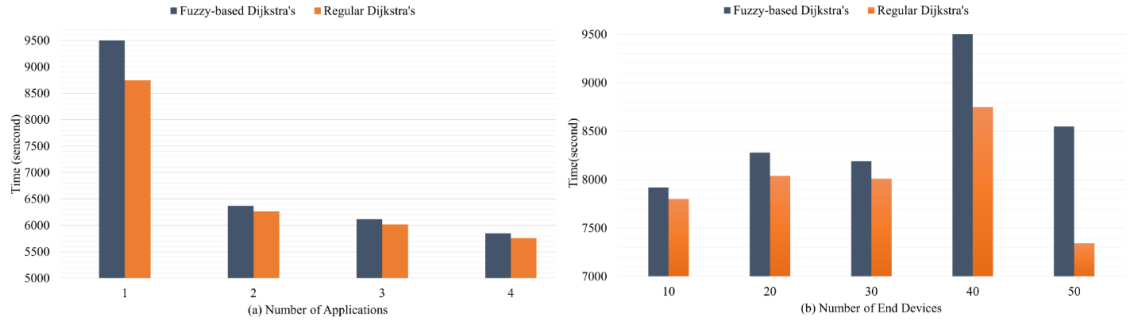


Figure 3.18. First ED deaths with number of EDs and number of applications

When the number of applications increases, the number of traffic sources increases and some intermediate EDs forward data to more applications, leading them to consume more energy. Moreover, every application has one path, so that when the number of applications increases, battery level decreases faster, especially for EDs shared in these paths, which die in a short time.

The proposed fuzzy-based Dijkstra's can find an alternative path after EDs' batteries have passed path recall threshold, to prolong network lifetime. Figure 3.18. illustrates a different number of EDs with one application, the ED dying after 7000 seconds of simulation time. Fuzzy-based Dijkstra's shows better results in all cases than regular Dijkstra's algorithm, because the path is selected based on rule base to re-prolong network lifetime. With 10, 20 and 30 EDs, the first ED dies due to cluster tree topology limitation, being the only path found in time. The advantage of an exchange is that the path can use a large number of ED batteries during the running time. Figure 3.19. outlines an example of system implementation; the study of SDNC routing algorithms in different clusters for the scenario is represented by one application and 50 EDs. The results display a comparison of the routing algorithms on the network in terms of number of ED deaths, delay, and throughput.

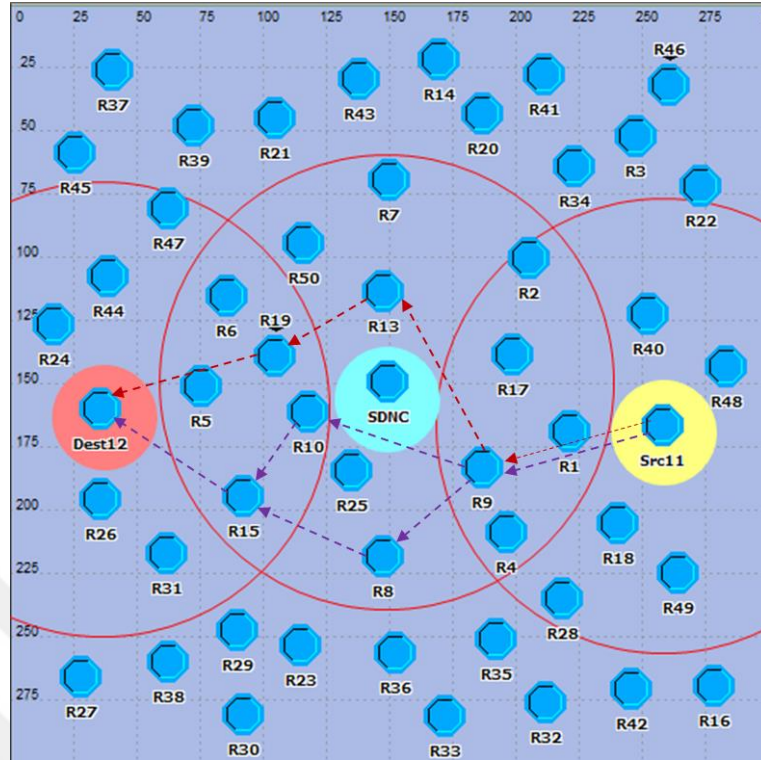


Figure 3.19. Scenario for 50 EDs and one application

Table 3.5. displays the paths for fuzzy-based Dijkstra's and for regular Dijkstra's. The simulation results indicate that regular Dijkstra's algorithm selects the shortest path according to the SNR as a cost, without considering the battery level of EDs. In fact, the EDs' batteries decrease over time. Therefore, fuzzy-based Dijkstra's selects paths using the fuzzy rule base and the consumed power of the EDs' batteries can be distributed among more EDs in the same time. Moreover, fuzzy-based Dijkstra's re-establish alternative path at 5089 seconds to prolong network lifetime as illustrated in Table 3.5.

Table 3.5. Path information

No of applications	Path establishment time, s	Path details	Total Cost
Fuzzy-based Dijkstra			
1	50	Src11=>R9 =>R10=>R15=>Dest12	16
1	5089	Src11=>R9 =>R8 =>R15=>Dest12	95
Regular Dijkstra			
1	50	Src11=>R9 =>R13 =>R19=>Dest12	273
ZigBee-based WSN			
1	50	Src2=>R1=>PAN coordinator=>R11=>14	-

The EDs in the path consume more energy than other EDs not involved in the application's path, therefore when EDs die over time, this can reduce the network lifetime. Moreover, other EDs in this topology remain live and their battery level remains higher than EDs in the path. For that reason, an alternative ED should be used during the transmission time to prolong network lifetime. Figure 3.20. displays the results when there is only one application running and there are 50 EDs in the network for aforementioned three models. The Figure shows that in a short time, the ZigBee-based WSN model contains a large number of death nodes more than the regular Dijkstra algorithm and the fuzzy based Dijkstra algorithm. Because there is no effective energy management strategy in the ZigBee-based WSN model.

Moreover, the regular Dijkstra algorithm over time has increases the number of death nodes more than the fuzzy based Dijkstra algorithm, as expected. Since the path selected by regular Dijkstra's remains fixed over time because the SNR values do not indicate which EDs die more quickly over time. During transmission time, the intermediate EDs in this path continue to forward data and consume much more energy, and other neighbour devices in the topology still have a high level of energy, so that the path ought to be reconfigured and reselected using the new costs.

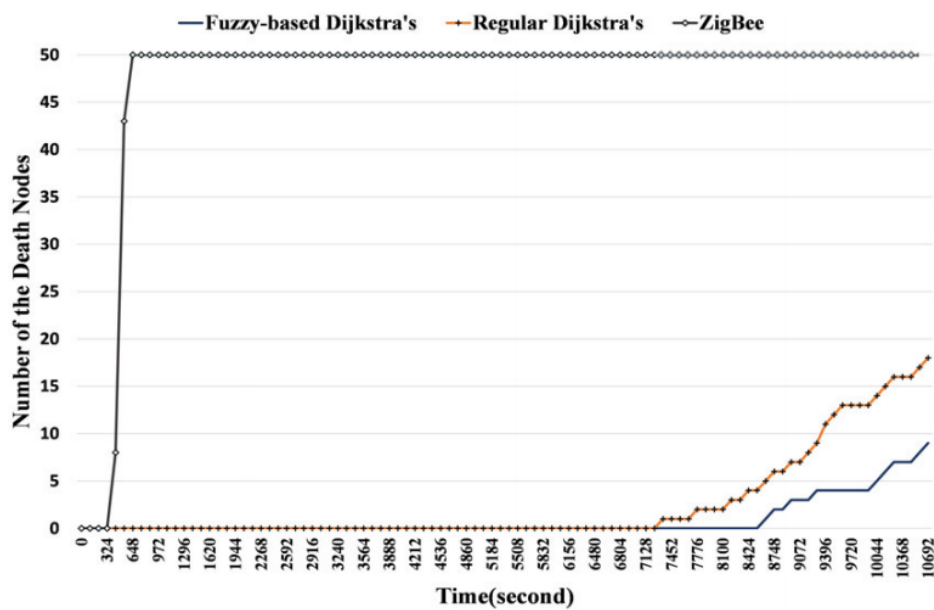


Figure 3.20. Number of ED deaths over time for 50 EDs and one application

Figure 3.21. exhibits network loads of intermediate nodes on the path. Some nodes that have a higher battery level take place instead of other nodes in the path. Therefore, the data transfer switches from node R10 to R8. this is because the SDNC rebuilds new path at 5089 s of the simulation runtime as shown in Table 3.5., to prolong the network lifetime.

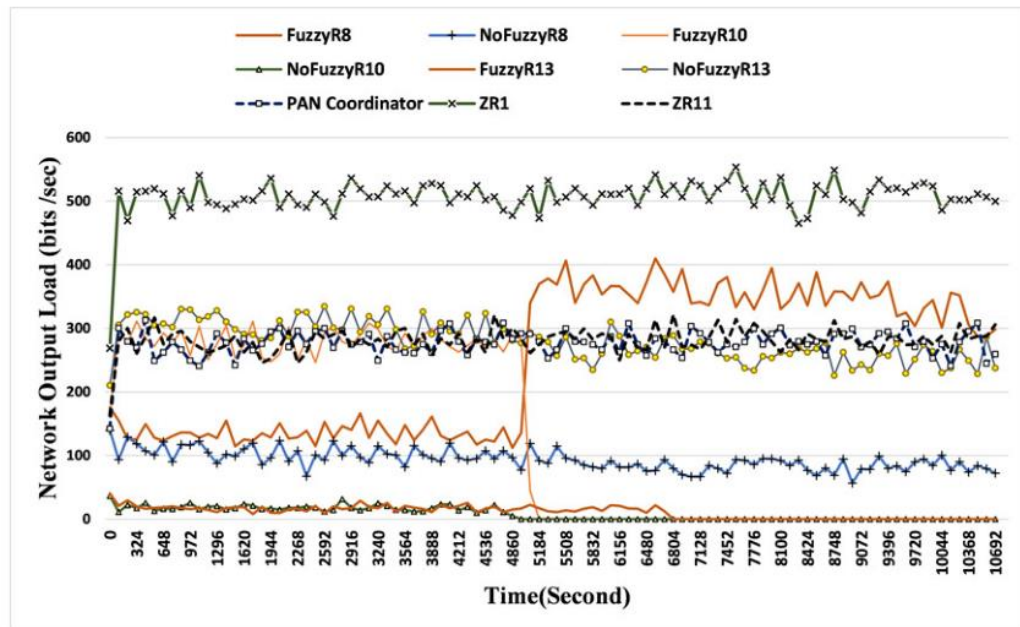


Figure 3.21. Network loads for the intermediate end devices for 50 EDs and one application

Fuzzy-based Dijkstra's reselects the path and creates a balance of lifetime battery consumption of the entire network for the devices that spend a great deal of energy, leading to sleep time to save energy. The throughput and end-to-end delay for 50 EDs are shown in Figure 3.22.; the expected results during prolonged network lifetime are the end-to-end delay test for 50 EDs.

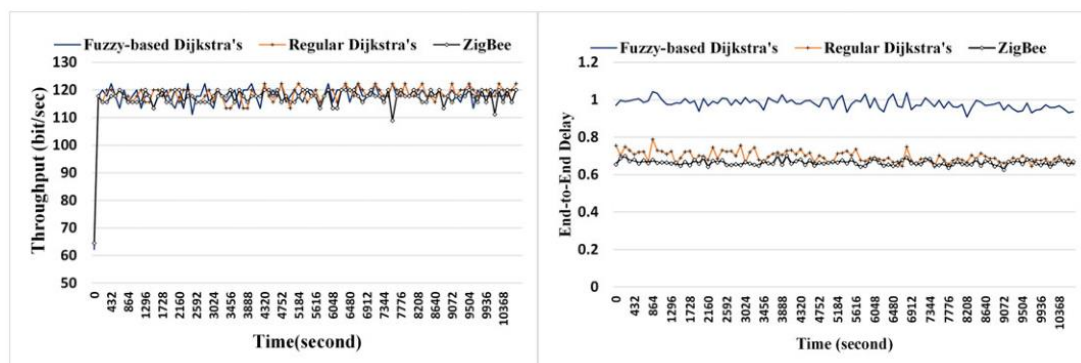


Figure 3.22. Throughput (bit/sec) and end-to-end delay for 50 EDs and one application

Thus, because fuzzy-based Dijkstra selects the best path according to the rule base which has a higher level of battery for related EDs, the delay increases. Regular Dijkstra selected the best path according to distance (SNR only) which decreases the end to end delay. Also, the ZigBee-based WSN has only one path that does not change over time similar to the Regular Dijkstra. This indicates the similarity of the end to end delay results. In Chapter 5 we develop the fuzzy-based Dijkstra to improve the delay results well.



CHAPTER 4. INTERACTING THE VERTICAL DOMAIN DEVICES OF IOT SYSTEM WITH HORIZONTAL DOMAIN SYSTEM

Hybrid systems require interaction between different IoT technologies to create intelligent services and smart environment. For the effective performance of IoT applications, IoT technology such as radio frequency identification (RFID) which is widely used to identify and track the location of RFID tags in various environments. For example, the healthcare system required a hybrid system to improve its services and access to healthcare information in real-time. In this chapter, we provide the communication between the vertical domain of IoT devices with the horizontal domain system. In the vertical domain system, the RFID technology is integrated into the SDN-based WSN as new platform to the healthcare system, the proposed is presented with simulation environment details and results.

4.1. IoT Technologies Related Work

Several projects also are proposed in the literature to integrate IoT technology such as RFID and facilitate interaction between the cloud and the vertical domain devices of IoT systems in real time. The following literature references on RFID are diverse; various solutions have been proposed to address the problems of RFID technology. Radio frequency identification technology is widely used in a large number of IoT applications; the authors in [53] propose combining smart parking solutions (SPS) with innovative IoT technologies such as RFID and WSN. The system collects real-time data on parking conditions and guides drivers to the nearest vacant parking space.

The IoT hybrid monitoring system presented in [54] proposes an RFID gateway for healthcare environments. The RFID gateway is designed for both WSN and RFID readers to transfer asset and patient data to the WSN network, using passive RFID tags

with sensors to track assets and patients in terms of location and condition. RFID technology has improved the cost effectiveness of the system and simplified the management of asset and patient data.

Radio-frequency identification and WSN integration into supply chains provides system intelligence, since WSN is multi-hop and RFID technology is single-hop [55]. An intelligent system is proposed in order to enhance system performance; a combination of RFID and sensors is applied. Radio-frequency identification is used for identifying tags and WSN is applied to sensing environmental variables such as humidity, temperature, and air quality. Some products should be protected against humidity and some must be kept at a certain temperature.

The paper [56] proposed a fire IoT service-oriented architecture consisting of four layers to connect fire tools to IoT networks. In the sensor layer, a smart sensor connected to the RFID reader/tags was provided to give warning data in order to send it to the base station. The RFID reader reads fire tools tagged with the RFID tag and can activate fire system devices such as water hoses to fight fires. The sensor senses data such as smoke, temperature, humidity and light, and uses fuzzy logic to detect the level of fire risk.

A security solution is proposed for IoT RFID Technology in [57]. The RFID authentication system includes a TSMMA PUF structure and bidirectional RFID authentication based on PUF. TSMCA PUF needs certain hardware resources to perform a four-way procedure between the arbitrator and the transmitter. The PUF works in two directions of authentication, from server to tag and from tag to server.

In [58], a framework is proposed to improve IoT applications' performance, and smart tag location tracking is applied to networks integrating both WSN and RFID. In this framework, the fuzzy q-algorithm and the fuzzy system-based route classifier are considered. The fuzzy q-algorithm is used to enhance the anti-collision protocol for RFID, and the fuzzy system-based route classifier to classify the paths and assist routing protocols.

The project described in [59] is an MQTT communication protocol and RFID technology in which a steel beam product has an EPC code in its RFID tag. It uses MQTT to send messages from RFID and IoT sensors. The RFID tag is tracked and sent from the warehouse using IoT sensors connected to the RFID/IoT system. This structure is extended to the IoT domain based on the EPCGlobal/GS1 framework.

The paper in [60] introduces the structure and components of a WSID heterogeneous network integrating WSN, RFID, and GIS technologies together. The platform uses four layers: (1) a detection layer including smart tags and RFID tags; (2) a monitoring layer including WSN gateways; (3) a management layer including a monitoring centre, data server, and application server; and (4) backup interfaces handling the interface between the WSID and external networks. The RFID tags in this work save information such as material category, serial number, personnel ID, and location, so that a different RFID reader in the system can collect it.

In [61], cybersecurity seeks to design smart and secure parking solutions based on technologies such as WSN, RFID, Ad-hoc network, and the IoT. They rely on RFID to determine vehicle registration numbers and related data including parking number, parking period, parking fee, and password assigned for security purposes.

4.2. Vertical Domain Devices

In this section, a new real-time IoT-based data analytics architecture for smart healthcare is proposed. The new platform includes the SDN-based WSN as explained in chapter 3 and RFID structure, which are the key contribution of the study in the vertical domain. The patient's related Electrocardiogram (ECG) data is sensed by the sensing nodes in WSN and delivered to the gateway (destination) node through the path established by SDN controller so that it could be employed in data analytics operations.

The RFID technology in the proposed structure is in charge of identifying the patients. We assume that each patient is given an RFID wristband. For more realistic performance evaluation all the vertical domain components have been modeled and

simulated using Riverbed Modeler. Since any IoT enabling technologies in the vertical domain can be modeled using a simulation software, the proposed architecture has a big potential to be used as a time-saving experimental environment and, this case can be considered as the other contribution of this study.

Using RFID tags with sensors to track assets and patients in terms of location and condition. RFID technology can improve the cost-effectiveness of the system and add simplicity to manage the data of assets and patients. The proposed real-time data analytics architecture for smart healthcare consists of two domains, namely the vertical domain (SDN-based WSN and an RFID structure), and a horizontal domain (Kafka, Spark, NodeJS, and Mongo database) as outlined in Figure 4.1. The patient-related ECG data is sensed by the source sensors (source SN) in the WSN and is delivered to the gateway so that the data can be transferred to the Kafka platform using the TCP socket. The Kafka messaging system distributes incoming data to three different consumers associated with it, as can be seen from Figure 4.1. The first Kafka consumer is the Spark platform for real-time data analysis. The second one is NodeJS web application that visualizes the patient's data in the web browser, and the last consumer is the MongoDB database that stores the incoming data for future usage.

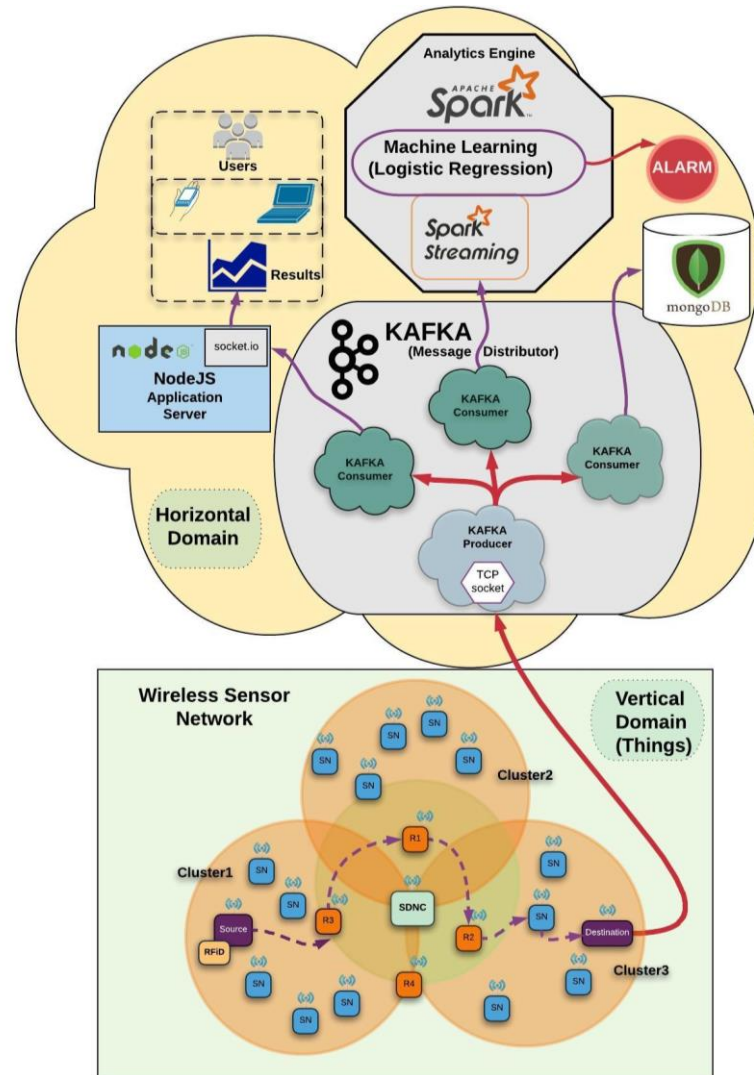


Figure 4.1. Outline of the RFID in vertical domain architecture.

4.2.1. Software-defined networking based wireless sensor network

As stated earlier in chapters 2 and 3, the SDN based WSAN is developed through the study given in [12]. In this study, each SDN-enabled sensor node (SN) is connected to the RFID reader and the patients' heart rate device to collect data. These sensors which are represented as source sensors then delivers the measured data to the gateway via the WSAN in real-time.

4.2.2. Radio frequency identification

RFID is a widely used IoT enabling technology for data collection. Figure 4.2. illustrates the Riverbed node model that integrates RFID reader into the SN. While the RFID related components of the vertical domain modelled here in this study, the SDN-based WSN part was modelled through the study given in chapter 2.

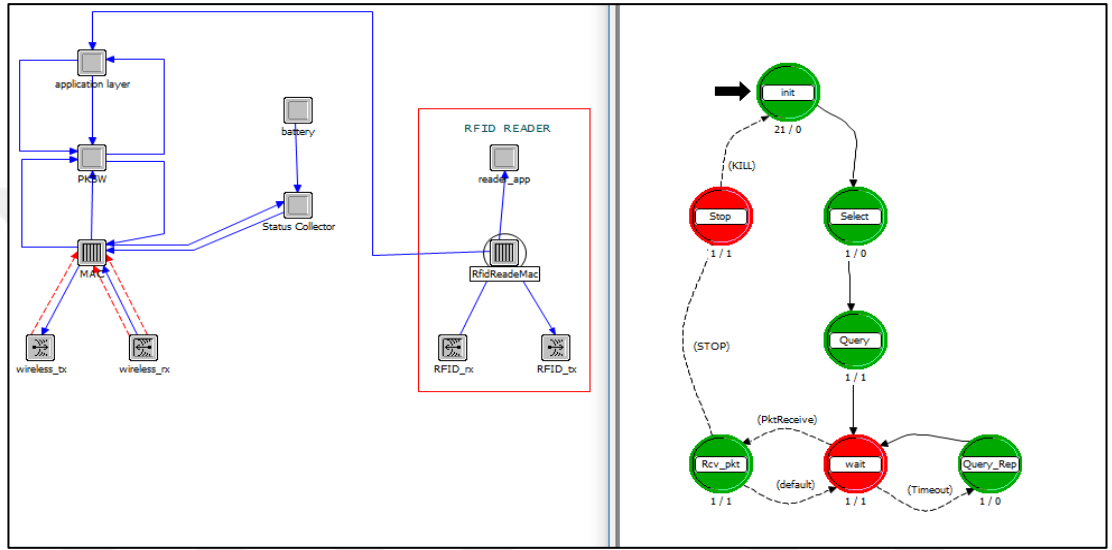


Figure 4.2. SN with RFID reader node model and RFID reader process model.

The followings are a brief explanation of the RFID reader module that is embedded into the SN:

1. The sensor application layer module was developed to receive patient data from the MAC module of the RFID reader and to integrate this data with patient PR data in the packet generator process.
2. The RFID MAC module was developed and presented in [62]. We have modified this MAC module to meet the proposed SDN-WSN design requirements. RFID reader can communicate with RFID Tag using the main functions which are:
 - a) ReaderSelect() is a function to select RFID Tag with number.
 - b) ReaderQuery() is a function to query a packet with a query value.
 - c) Query_rep() is a function to create a “queryAdjust” packet format and create a “queryReply” packet format as requested by an RFID tag.

- d) Reader_Received_Packet() is a function that is used when the packet is received for the ACK packet, which will be generated and replied to each valid type of packet: RN16, EPC, XPC, CRC16, and patient data.
- e) Reader_kill() is a function to kill the packet when the session ends.
3. The RFID reader's application module collects statistical information about the RFID tag.

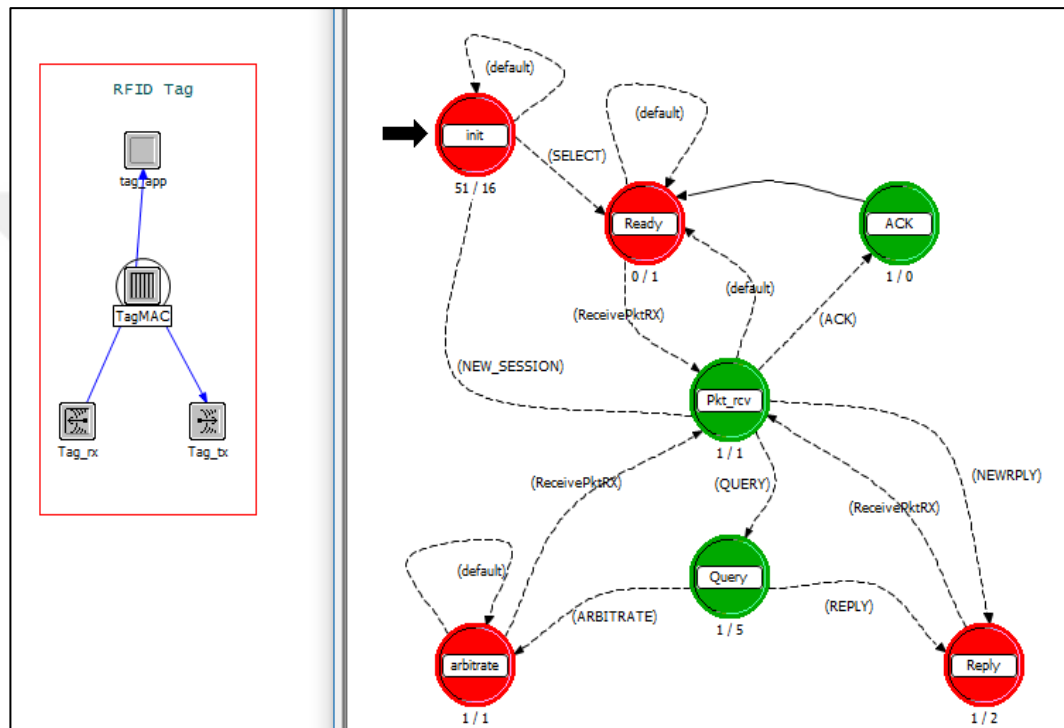


Figure 4.3. RFID tag node model and process model.

The RFID tag node model contains two modules [62] as illustrated in Figure 4.3. A brief explanation of the model is given below:

1. The MAC module is developed for the RFID tag. This MAC module collects patient data for sending to an RFID reader. In this model, the main functions to communicate with an RFID reader are:
 - a) Receive_packet() is a function to receive and respond to any type of RFID reader command packets.
 - b) QueryTag() is a function that responds to the query for any type of packets in the case of a tag query.

- c) ReplyTag() is a function in the reply state to create a 16-bit random number (RN16) packet and send it to an RFID reader.
 - d) AckTag() is a function in the case of the ACK's state to receive an acknowledged packet from an RFID reader.
- 2.The RFID Tag's application module collects statistical information about the patient's data.

4.3. Horizontal Domain System

Nowadays, the growing volume of data on the internet has also increased the importance of big data analytics, which makes sense from this data. It is obvious that this large amount of data will increase further with the widespread use of the internet of things which would incorporate numerous devices into the existing internet infrastructure. In the horizontal domain system, the platform developed in this study has a real data analysis structure. Therefore, the proposed platform contains high-scaled and high-performance data analysis tools such as Kafka, NodeJS, Spark, and MongoDB in the horizontal domain. The components used in this domain are; i) a Kafka messaging system that distributes incoming data to different consumers, ii) a web application which was developed using NodeJS to visualize the patient data, in real-time, iii) a Spark platform for real-time data analytics, and v) a NoSQL database that stores incoming data.

4.3.1. Kafka

Kafka (version 2.11) [63] is the actual distribution system, a high-throughput distributor for messages, dealing with the enormous amount of data and supporting a huge number of consumers and producers. Kafka uses a number of partitions and a number of brokers to perform parallelism. The parallelism accelerates the processes effectively. In addition, it automatically retrieves data in cases where the broker fails. Through these characteristics of Kafka, the real-time data streaming requirements is supported.

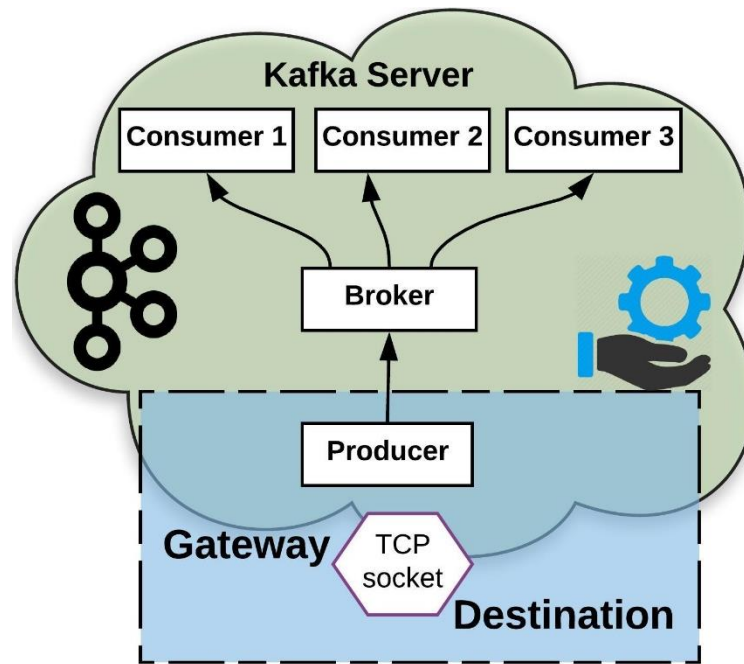


Figure 4.4. Streaming gateway and Kafka (producer/consumer).

Figure 4.4. shows the system gateway (destination SN) associated with the Kafka producer, through this portal, the flow of data directed from the vertical domain in real-time to the horizontal domain. This gateway is a part of the vertical domain system and is connected to Apache Kafka server using a TCP socket server program. Using TCP socket program (server and client connection) makes communication reliable and oriented. Kafka producer receives data stream from the TCP client sockets program, which in turn listens to a particular port and IP address. Thereafter, Kafka producer sends the streaming data to Kafka's consumers.

4.3.1.1. Kafka Server and Kafka Client Setup

Kafka server and client are connecting from different platform using TCP socket client/server programs. Kafka producer is using java-based socket program while Kafka producer is using c-based socket program. Figure 4.5. shown the start running steps of zookeeper and Kafka servers.


```

C:\kafka_2.11-0.11.0.1\bin\windows>zookeeper-server-start.bat ..\..\config\zookeeper.properties
[2017-11-30 13:45:56,196] INFO Reading configuration from: ..\..\config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

C:\kafka_2.11-0.11.0.1\bin\windows>kafka-server-start.bat ..\..\config\server.properties

```

Figure 4.5. The commands to run zookeeper and Kafka servers

Riverbed node model connects to the Kafka server using TCP socket client/server programs. The following steps include setting up Kafka to create a topic as well as the connections between Riverbed node model and the Kafka server.

- 1.To create a topic with name "testing1" in apache Kafka use the following command:
`./kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic testing1`
- 2.When Riverbed simulation starts working, the source SN generates data to send it to the destination SN. Riverbed node model includes the c-based socket file in header of the SN application layer as illustrated in Figure 4.6.

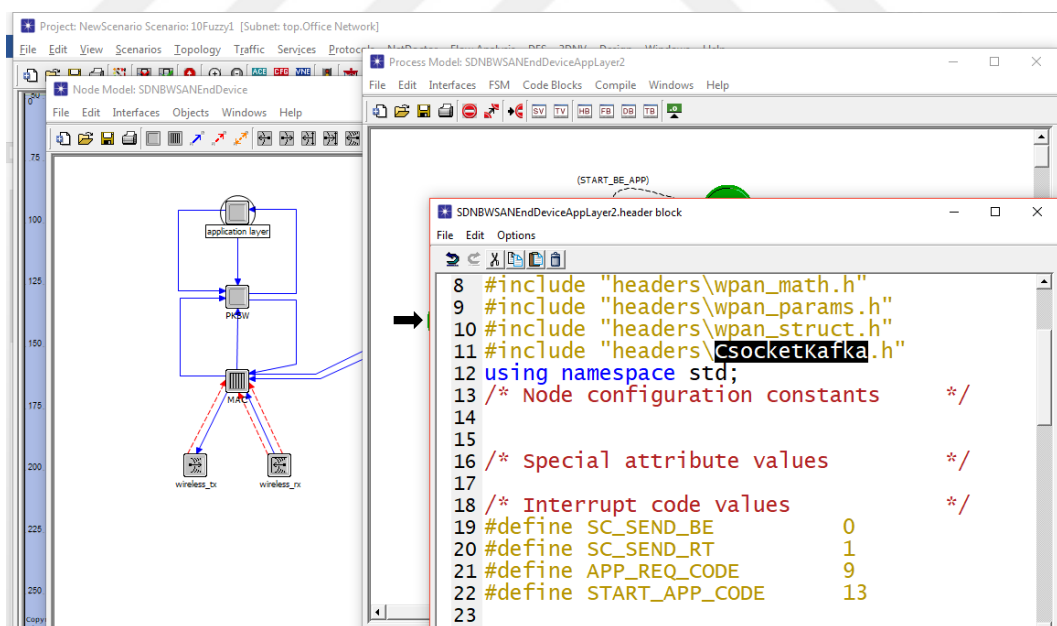


Figure 4.6. Include the c-based socket in riverbed node model

- 3.The destination SN receives data packets and executes c-based socket client program by using the script to send number of messages to the Kafka server as shown in Figure 4.7.

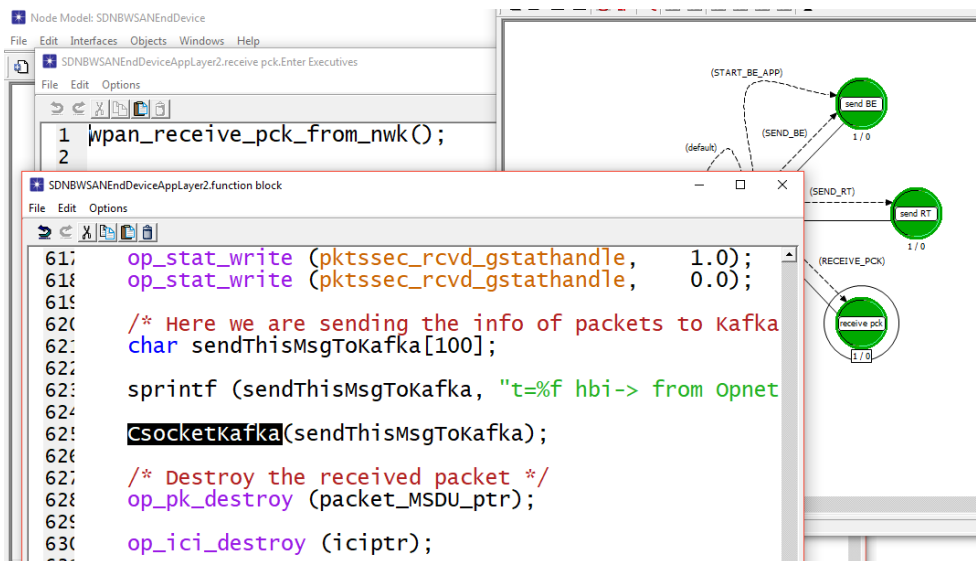


Figure 4.7. The destination SN (gateway) use the socket to send packet to Kafka

4. The java-based server socket gets these messages from c-based client socket to Kafka producer as shown in Figure 4.8. Meanwhile, The Kafka producer sends messages to Kafka consumer.

```

29
30 try (Socket clientSocket = serverSocket.accept())
31 {
32     debug(clientSocket.getInetAddress().getHostName() + " : " + clientSocket.getPort() + " Co
33     debug(clientSocket.getOutputStream() + " : " + clientSocket.getPort());
34     debug(clientSocket.getInputStream() + " : " + clientSocket.getPort());
35     out = new PrintWriter(clientSocket.getOutputStream(), true);
36     Date now = new Date();
37     out.write(now.toString() + "\r\n");
38     out.write("This Message from server" + "\r\n");
39     debug("After I write Message in the client side ");
40     BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream
41     String textFromClient = null;
42     String textToClient = null;
43     textFromClient = in.readLine(); // read the text from client
44     out.print(textToClient + "\r\n"); // send the response to client
45     debug(textFromClient + "|| client output ...");
46     kafkaConnection(textFromClient);

```

Problems @ Javadoc Declaration Console Properties

JavaKafka [Java Application] C:\Program Files\Java\jre1.8.0_152\bin\javaw.exe (Dec 1, 2017, 6:47:30 PM)

Client: java.net.SocketInputStream@3ba987b8 : 51277

Client: t=686.350368 hbi-> from Opnet send this MSG to Kafka [Dest12 (#12)] packets=319 || client output

Client: 0:0:0:0:0:0:1 : 51280 Connected.

Client: java.net.SocketOutputStream@3f191845 : 51280

Client: java.net.SocketInputStream@5f049ea1 : 51280

Client: t=688.315808 hbi-> from Opnet send this MSG to Kafka [Dest12 (#12)] packets=320 || client output

Client: 0:0:0:0:0:0:1 : 51283 Connected.

Client: java.net.SocketOutputStream@72cc7e6f : 51283

Client: java.net.SocketInputStream@5afa3c9 : 51283

Client: t=690.404448 hbi-> from Opnet send this MSG to Kafka [Dest12 (#12)] packets=321 || client output

Client: 0:0:0:0:0:0:1 : 51286 Connected.

Client: java.net.SocketOutputStream@72835809 : 51286

Client: java.net.SocketInputStream@909217e : 51286

Client: After I write Message in the client side

Figure 4.8. Java based socket

4.3.2. Apache spark

Apache spark (version 2.11) [64] is an open source computing platform developed using scala, which enables parallel processing of large data sets. Because apache spark's operations are performed in memory, it supports data analysis with high performance and high speed for various formats (static or streaming). Moreover, it includes machine learning libraries.

4.3.2.1. Apache spark streaming

Apache spark processes real-time streaming data very quickly because of maximum efficiency during operations, therefore, it processes a large amount of data in a short time.

4.3.2.2. Apache spark machine learning library

Machine learning library (MLlib) is scalable library in the apache spark that contains common machine learning algorithms, including classification, regression, and aggregation. The logistic regression algorithm is one of the regression machine learning algorithms, which is selected in this study to classify patient data and provide better results for IoT healthcare applications.

4.4. Case Study: Real-Time Disease Diagnosis Using Logistic Regression

Real-time diagnosis is a new and important way to diagnose a patient's condition. In order to perform real-time disease diagnosis, we used a logistic regression algorithm that is machine learning technique in apache spark to predict patient's condition.

4.4.1. Electrocardiogram

ECG refers mainly to the electrical activity of the heart. ECG consists of 5 waves: P, Q, R, S, T corresponding to different stages. ECG allows us to record the activities of

the heart muscle from several points on the surface of the body [65]. In this case, ECG plays an important role in the diagnosis of various heart diseases. In our study we considered about QRS and PR intervals which are related to Wolff Parkinson white (WPW) disease.

Wolff Parkinson white syndrome is a conduction disorder where the atrial pulse passes to the ventricle through an extension path along the normal ventricular atrium junction. Patients with WPW may suffer from palpitations, fainting, and sudden death [66]. The symptoms of this disease are a PR interval of less than 120 (shorter than 0.12 seconds) and a QRS compound that is greater than 110 (greater than 0.11 seconds). In order to assess and classify WPW disease, the QRS and PR intervals are selected from the arrhythmia dataset in the UCI database [67]. This dataset consists of the ECG data from 452 people, each with 279 attributes.

4.5. Experimental Results and Discussions

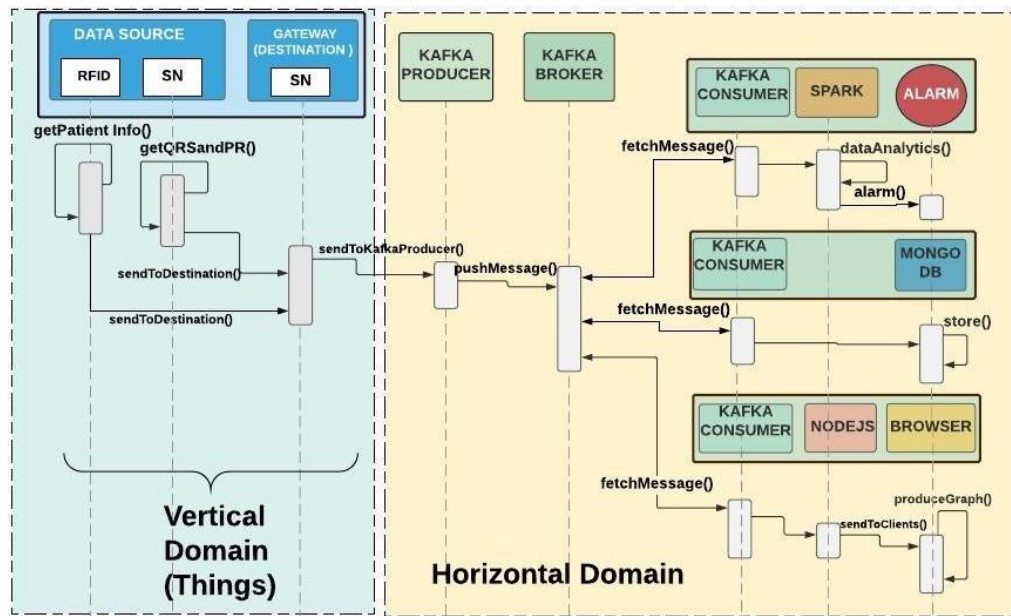


Figure 4.9. The Sequence diagram for end-to-end data delivery operations.

The sequence diagram shown in Figure 4.9. presents the end-to-end data delivery operation of the proposed system. It clarifies the main operations performed to transfer the patient's data from the vertical domain to the horizontal domain. The vertical

domain that has been modelled and simulated using Riverbed Modeler has source SNs, RFIDs and a gateway, as can be seen from the figure. On the other hand, the horizontal domain of the proposed architecture has been built using real-world technologies such as Kafka, Spark, NodeJS and MongoDB. The simulation of the WSN and RFID was run for 3600 seconds. The source SN generates QRS and PR data of any patient once in every 2 s during the simulation run time. This data is read from a file including test data. In the meantime, RFID reader that is embedded into the SN gets other related information from the RFID tag like tagID, patientID, age, gender, height (cm), and weight (kg). All of these pieces are combined and then, forwarded to the gateway using SDN-based WSN. The gateway delivers this data to the Kafka Producer to be distributed to the related Kafka Consumers. The communication between WSN (gateway) and Kafka (Kafka Producer) is provided using TCP/IP TCP sockets. As soon as the Kafka Producer pushes incoming data into the Kafka Broker, all Kafka Consumers fetch it for use in their individual job.

As can be seen from Figure 4.9., the Kafka consumers are connected to the web application, Spark engine, and MongoDB, respectively. When Apache Spark gets the patient related ECG data, it tries to diagnosis disease using Logistic Regression model, in real-time. NodeJS web application sends the data from Kafka instantly to the connected client browsers using socket.io module. The illustration of the web-based user interface in Figure 4.10. shows the last record of patient data that is delivered by NodeJS application. Finally, all data introduced to the cloud is stored in the high-performance MongoDB database for future use.

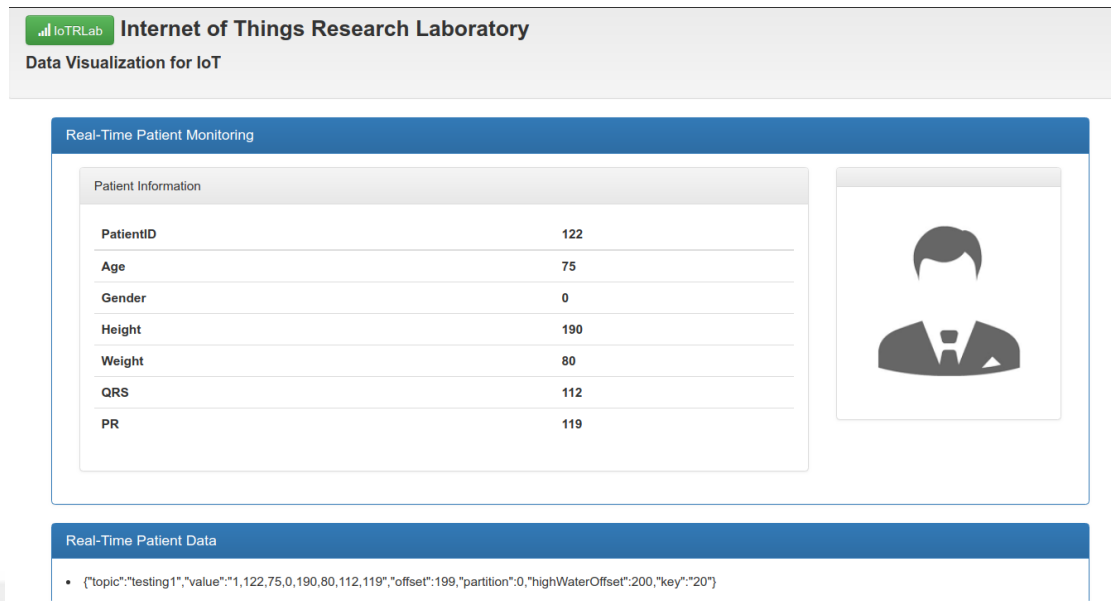


Figure 4.10. NodeJS patient information interface.

4.5.1. Simulation environment and hypothesis

In this study, the simulation scenario is examined to look into the performances of the developed models using the simulation model in [12]. An explanation of the network, together with the developed components, is shown in Figure 4.11.



Figure 4.11. Streaming scenario for IoT healthcare applications.

Figure 4.11. shows the simulation streaming scenario; patient data comes from three RFID tags and three sensor sources. Each RFID tag is associated with SN to collect patient information such as patient identification, age, gender, height (cm) and weight (kg). SN source generates patient's data such as QRS and PR as test data.

4.5.1.1. Numerical results and argumentations

In this part, the composition of the logistic regression algorithm tested under the parameters as shown in Table 4.1. In Table 4.2., we can see the classification type for the diagnosis of WPW by a logistic regression algorithm. Training data is the correct data that logistic regression algorithm learned and build model. Testing data is the streaming data which generated randomly from the vertical domain system as shown in Table 4.3. These data are evaluated in real-time and the logistic regression algorithm predicted the results for each group of patients depicted as shown in Figure 4.12. The total average of the data (Input Rate) in Kafka/Spark streaming processes tested and

investigated in different partitions as shown in Table 4.4. Finally, the proposed system can deal with unexpected failures using number of partitions as explained in Table 4.5.

Table 4.1. Logistic regression algorithm parameters.

Parameter	Value
set max iterations	10
set regularization parameter	0.3
set elasticnet mixing parameter	0.8

Table 4.1. shows the parameters used in the logistic regression algorithm settings. Max Iterations is the maximum number of runs. The Regularization parameter is an input value of lambda in the model, which reduces the variance of the estimated regression parameters. The regularization path is computed for the elasticnet penalty at a grid of values for the regularization parameter lambda.

Table 4.2. Classification table.

Class Number	Patient Sate
0	Not Wolf Parkinson White
1	Wolf Parkinson White

Table 4.2. shows the data classification of the logistic regression algorithm, which classified the data into two categories of WPW disease.

Table 4.3. Data streaming.

Data	Record Numbers
Training Data	350
Testing Data	15000

In Table 4.3., the logistic regression builds the trainer model from 350 patients as training data. The algorithm was also tested on 15,000 data streaming of patient records as testing data.

Column	PatientID	Age	Gender	Height	Weight	QRS	P-R	features	label	rawPrediction	probability	prediction
1	1423	55	0	168	64	52	59	[52.0, 59.0]	0.0	[0.49163807170775...	[0.62049224418433...	0.0
2	335	45	1	146	69	124	129	[124.0, 129.0]	0.0	[0.35863812706202...	[0.58871072309332...	0.0
3	1111	22	0	160	65	34	85	[34.0, 85.0]	0.0	[0.86381485878467...	[0.70345707098266...	0.0
4	1423	55	0	168	64	101	143	[101.0, 143.0]	0.0	[0.68442893466572...	[0.66472647179666...	0.0
5	335	45	1	146	69	86	33	[86.0, 33.0]	0.0	[0.0312941363457...	[0.49217710433158...	1.0
6	1111	22	0	160	65	19	27	[19.0, 27.0]	0.0	[0.55324566472960...	[0.63488828168387...	0.0
7	1423	55	0	168	64	86	135	[86.0, 135.0]	0.0	[0.76343077614570...	[0.68209812854168...	0.0
8	335	45	1	146	69	146	138	[146.0, 138.0]	0.0	[0.22147220961547...	[0.55514284036741...	0.0
9	1111	22	0	160	65	98	15	[98.0, 15.0]	0.0	[0.2846062748708...	[0.42932485005444...	1.0
10	1423	55	0	168	64	31	106	[31.0, 106.0]	1.0	[1.05570133514251...	[0.74186820823050...	0.0
11	335	45	1	146	69	95	3	[95.0, 3.0]	1.0	[0.3498366819661...	[0.41342202584060...	1.0
12	1111	22	0	160	65	65	71	[65.0, 71.0]	1.0	[0.46264634069266...	[0.61364177364493...	0.0
13	1423	55	0	168	64	4	50	[4.0, 50.0]	0.0	[0.87378154824131...	[0.70553194983074...	0.0
14	335	45	1	146	69	132	128	[132.0, 128.0]	0.0	[0.27546899586301...	[0.56843503966074...	0.0
15	1111	22	0	160	65	52	4	[52.0, 4.0]	0.0	[0.06310993261921...	[0.51577224860936...	0.0
16	1423	55	0	168	64	127	108	[127.0, 108.0]	0.0	[0.16675165070418...	[0.54159158229028...	0.0
17	335	45	1	146	69	70	142	[70.0, 142.0]	0.0	[0.96872614209722...	[0.72486551808961...	0.0
18	1111	22	0	160	65	43	118	[43.0, 118.0]	0.0	[1.03613181793845...	[0.73810294845955...	0.0
19	1423	55	0	168	64	66	75	[66.0, 75.0]	0.0	[0.48438980972442...	[0.61878392722149...	0.0
20	335	45	1	146	69	52	128	[52.0, 128.0]	0.0	[1.02924610074611...	[0.73676971063467...	0.0

only showing top 20 rows

Test Error = 0.29850746268656714
Model Accuracy = 0.7014925373134329

Figure 4.12. Logistic regression predicted results.

Figure 4.12. displays patients' incoming data, as well as the logistic regression algorithm's classification of the data in real time and the predicted results according to the data training model. Patient data are patientID, age, gender (0 for males and 1 for females), height, weight, QRS, and PR. The 'label' column is the estimated class rating. For example, for patientID (335) in the second row, which shows a QRS interval of 124 milliseconds and a PR interval of 129 milliseconds, the label rating value is equal to the expected value of the logistic regression. In this case, the logistic regression classifies this record as negative for WPW disease (patient = 1, healthy = 0). When the data for this patientID (335) changes to a QRS interval of 86 milliseconds and a PR interval of 33 milliseconds, as seen in the fifth row of Figure 4.12., the label value is not equal to the expected logistic regression. In this case, the logistic regression classifies this record as positive for WPW disease. The accuracy of the model resulting from this set of records is about 70%.

Table 4.4. Total average of the input rate.

Streaming	Partition 1	Partition 2	Partition 3
Input Rate	123.25 records /sec	140.10 records /sec	165.18 records/sec

Table 4.4. shows the total average of streaming data which is tested in three topics with different number of partitions. Each topic of Kafka distributes the incoming data received across a number of partitions that is created with this topic. The Kafka topic

receives data across a distributed set of partitions. As can be seen in Table 4.4., when the number of partitions increases, the number of records also increases.

Table 4.5. Streaming Data with Number of Partitions.

Input data size	Partition	Offset 1	Offset 2	Offset 3
120 records	1	16552 to 16671	-	-
120 records	2	10300 to 10365	10182 to 10237	-
120 records	3	34506 to 34548	34668 to 34707	34503 to 34542

The Kafka's offsets management help to restore the state of the stream throughout Spark direct streaming's lifecycle and deal with unexpected failures. Table 4.5. indicates the input data size of 120 records, which contains the offset numbers for each partition. When the number of partitions in the Kafka data streaming was increased, the number of offsets was increased; for example, partition 1 included one offset, whereas partition 2 included two groups of offsets and partition 3 included three groups of offsets. The benefit of increasing the number of partitions is if one of the partition's offsets fails, Kafka continues processing by using the other offsets.

We tested several scenarios by setting the number of patients in each application where is one application includes the source SN and RFID reader. First scenario with one application and second with two applications and the third with three applications, each application include two patients. The end to end delay and throughput results for three scenarios are shown in Figure 4.13. and Figure 4.14. respectively.

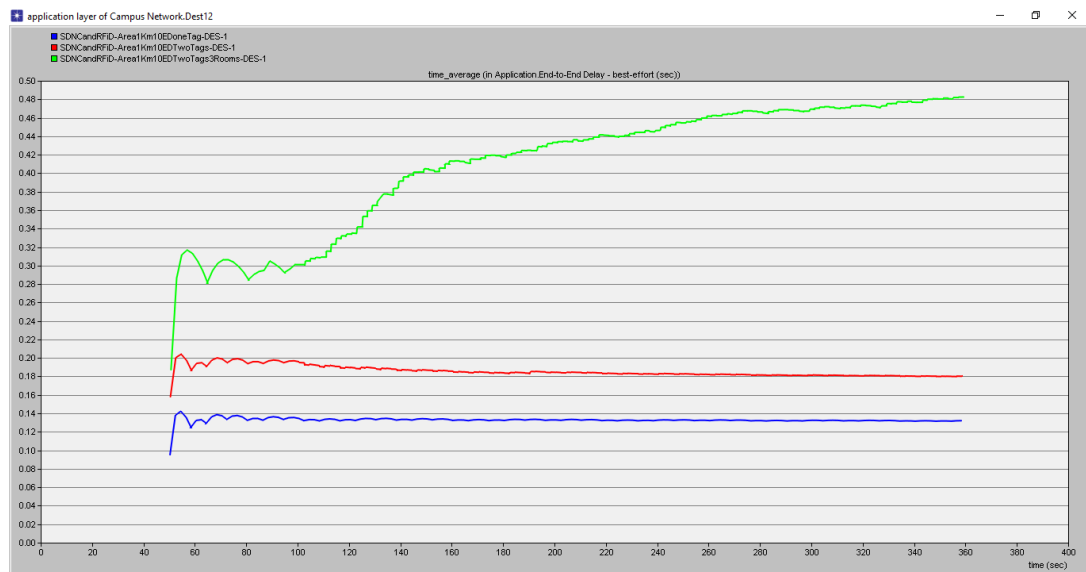


Figure 4.13. The end to end delay for the three applications

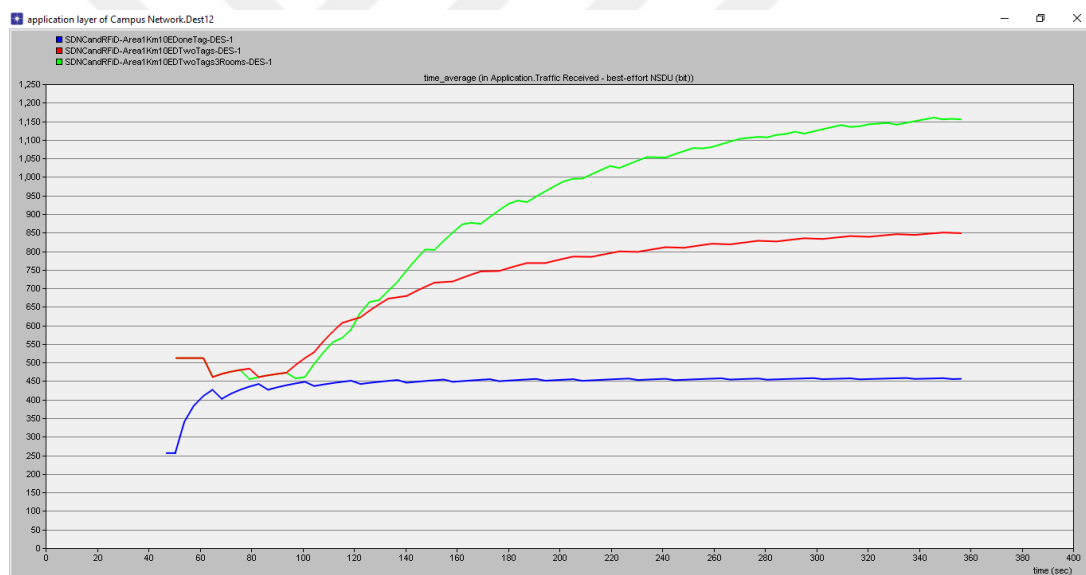


Figure 4.14. Throughput (bit/s) for the three applications

It is possible that the path is changing during transmission time using the SDN based WSN to prolong network lifetime as discussed in previous chapters. We can be observed from the figures when number of applications increased lead to a higher end to end delay, as well as the throughput results.

CHAPTER 5. ENHANCED SDN TOPOLOGY DISCOVERY AND IOT GATEWAY

5.1. Introduction

The beacon frame is the key factor to configure the IEEE 802.15.4 superframe. The components of the IEEE 802.15.4 superframe structure were discussed in chapter 2. The beacon settings have a power of influencing the WSN performance such as throughput and delay which are essential elements to meet the requirements of IoT applications. The SDN network topology discussed in chapter 3, which involved the topology discovery process to configure the beacon frame.

In this chapter, we improve the topology discovery process in a way that relies on the resulting path of the Fuzzy-based Dijkstra algorithm. This method is called an Enhanced SDN-WSN Fuzzy-based Dijkstra (ESWFD).

Another issue is to connect the system to the cloud via an IoT gateway. As we have seen in chapter 4, we provided the communication between the vertical domain of IoT devices with the horizontal domain system using the gateway. The system gateway directed the data flow in real-time from the vertical domain to the horizontal domain. This gateway is part of a vertical domain system that uses a single client / server TCP socket program. In this chapter, we enhance the system gateway using message queue telemetry transport (MQTT) protocol. The MQTT is a simple messaging protocol with low bandwidth to connect sensors (resource-constrained) devices to the internet. In MQTT, we can create many topics and from any system, many senders and receivers can share either a single topic or multiple topics.

5.1.1. Related research

The following literature references on the MQTT protocol are diverse. An application programming interfaces of MQTT and API web for IoT cloud is presented in [68] which the results are evaluated in term of average response time, average transmission delay and memory occupation for many tasks of IoT applications. An application layer protocols such as MQTT and CoAP protocols are analysed with respect to the cloud in [69] which their IoT architectures consists of 6 levels. The authors of [70] propose a real-time problem-oriented solution through IoT technologies and vehicle cloud service, therefore, the cloud parking service is designed according to the MQTT communication principle. The implementation of MQTT-S tested on the IBM wireless sensor networking testbed [71], which is integrated to ZigBee-based networks.

In this chapter, the main contributions can be summarized as follows:

1. To enhance Fuzzy-based Dijkstra's algorithm in SDN beacons structures via cross-layer (Topology Discovery and MAC layers).
2. Develop an interface between MQTT [72] and Riverbed Modeler simulation allowing the system to send WSN data to IoT cloud.
3. Develop a IoT web interface using NodeJS [73] for monitoring application to receive the network data from SDN and application data from destination node.

5.2. The Proposed ESWFD for IoT Applications

The proposed ESWFD architecture consists of two domains, namely the vertical domain (SDN and WSN structure), and a horizontal domain (MQTT, and NodeJS) as outlined in Figure 5.1.

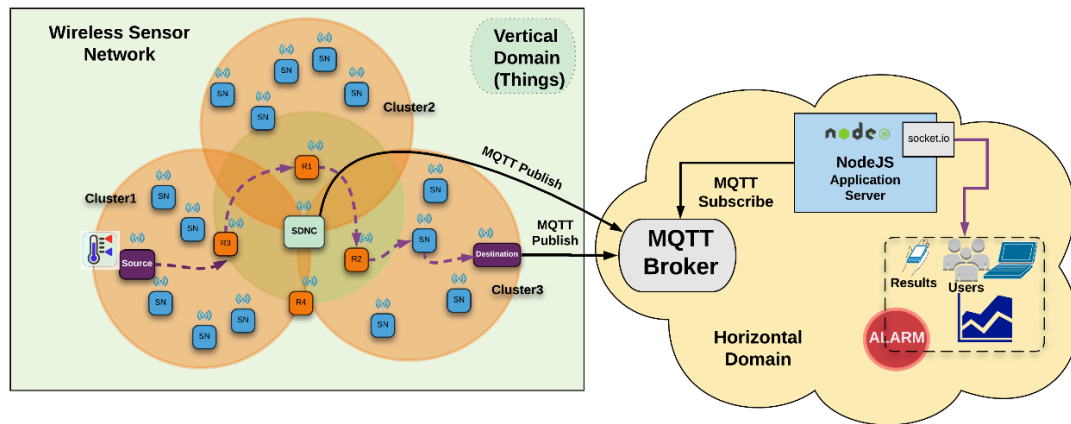


Figure 5.1. The general description of the ESWFD proposed IoT architecture.

5.2.1. Components of the ESWFD system

As stated earlier, the SDN-based WSN is developed through the study given in [12] and chapter 3. The new components of SDN node model as shown in Figure 5.2. (a), which consist of four main modules as the following:

1. The MAC module [12]. This module modified to reconfigure beacon according to the topology discovery.
2. The Routing Decision and Flow Table (RDFT) module modified to allow the SDN controller to send node status information and flow table information to be monitored over the internet using MQTT broker.
3. The topology discovery module includes new procedures to configure SDN beacon frame.
4. The MQTT publish module was developed to send data to MQTT broker.

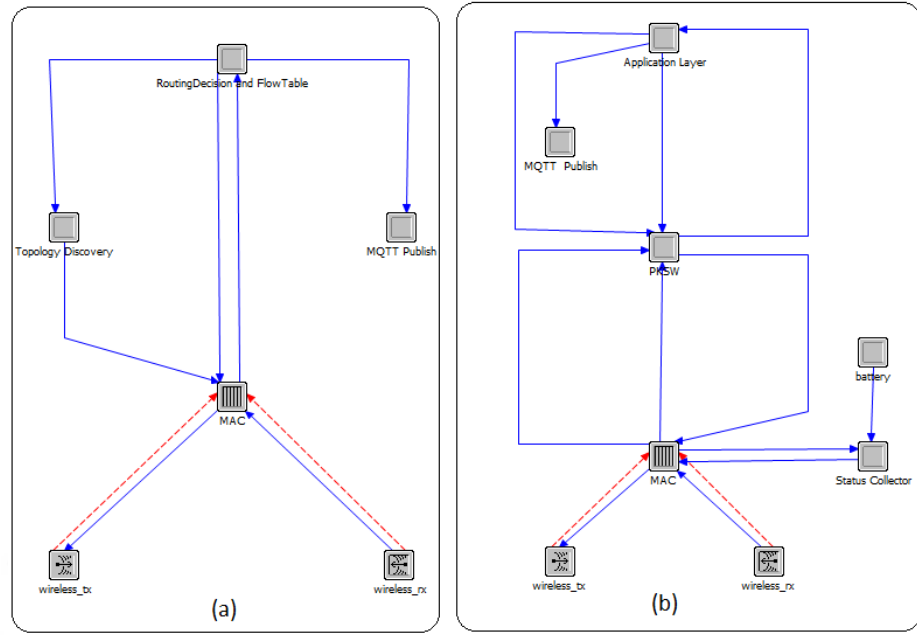


Figure 5.2. The system components, the node model of SDN (a) and the node model of sensor node (b).

In Figure 5.2. (b), the components of sensor node model consist of six modules, in this chapter, the new and modified modules are explained as the following:

1. The application layer module is for both source and destination nodes. Source node uses this module to generate data such as the temperature, humidity and carbon dioxide (CO₂) [74] data and sends it to the Packet Switch (PKSW) module. When this data arrived at this module, destination node (gateway) sends it to IoT could via his MQTT publish module.
2. The MQTT publish module was developed to send data to MQTT broker.

5.2.2. SDN topology discovery

Topology discovery in SDN supports a cluster topology for the active routing nodes, which every node transmits beacons utilized in the MAC and the Physical layers. In order to save energy and establish a synchronized network, the IEEE 802.15.4 beacon-enable mode is considered in [12]. Beacon-enable mode has active and inactive periods in the superframe [4]. The IEEE 802.15.4 standard active Superframe Duration (SD) [75] is defined in (Equation 5.1).

$$SD = aBaseSuperframeDuration * (2)^{SO} \quad (5.1)$$

The `aBaseSuperframeDuration` indicates the minimum duration of the superframe, corresponding to superframe order (SO) when it is equal zero. The SD active period should change the SD length to the number of routers in each cluster, especially when the application path is established between routers. This path aims to transfer data between number of clusters from source to destination nodes.

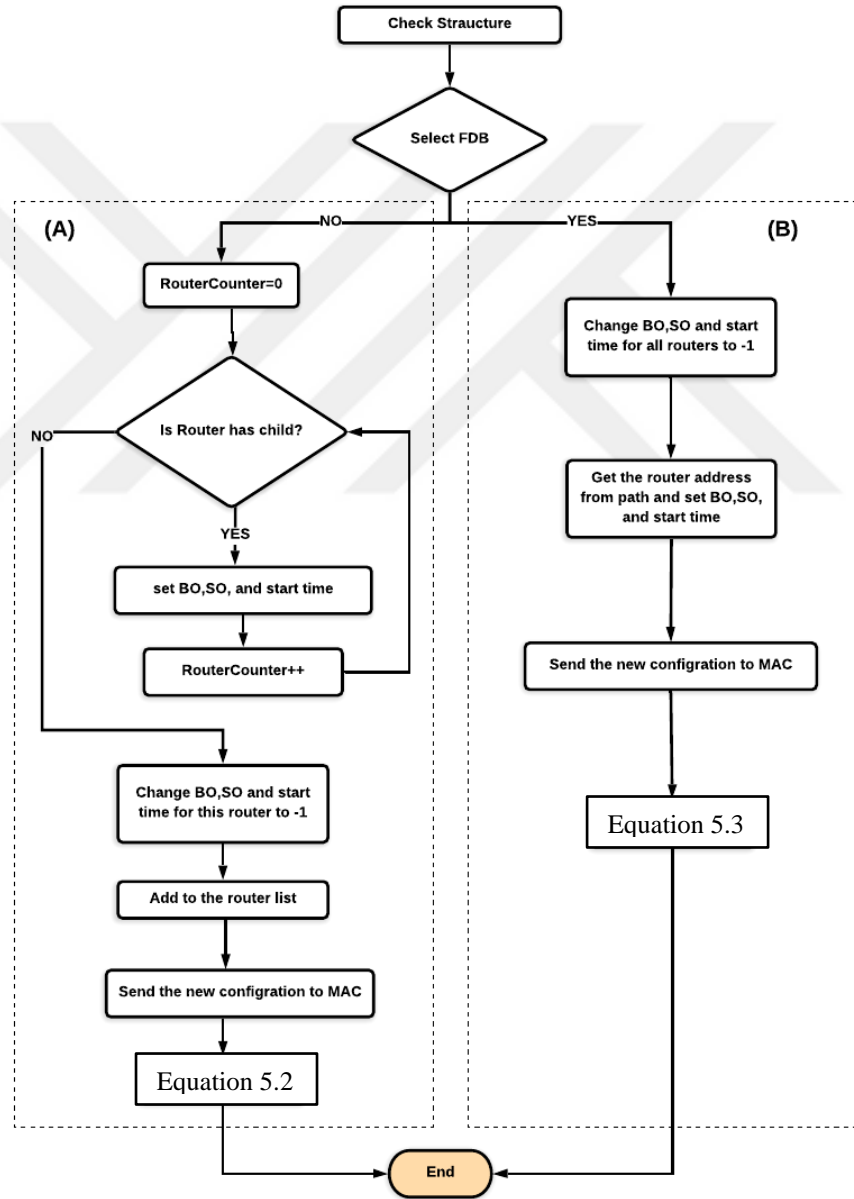


Figure 5.3. Flowchart of two algorithms to calculate the SD, Fuzzy-based Dijkstra (A) and Fuzzy-based Dijkstra Beacon (B).

The topology discovery procedures as shown in Figure 5.3. has two methods to calculate the SD in order to improve the network performance. The first method is the Fuzzy-based Dijkstra's (FD) [12] shown in Figure 5.3. (A), is checking the network structures according to the number of routers (NR) that have children nodes. In this method, the SD active period calculated in (Equation 5.2) according to NR and multiplied by (Equation 5.1). The second method is a Fuzzy-based Dijkstra's Beacon (FDB) shown in Figure 5.3. (B). This new method calculates the SD active period in (Equation 5.3), according to the cluster number (CN) divided by the routers counter (RC) that are in the application's path. Moreover, the beacon configuration of ZigBee-based WSN is shown in (Equation 5.1), which is the default setting for IEEE 802.15.4 standard. The Beacon's option of the three methods summarized in Table 5.1.

Table 5.1. Algorithms Configurations

No	Name	Beacon's option
1	Fuzzy-based Dijkstra's Beacon (FDB)	Cluster Number/ Routers Counter
2	Fuzzy-based Dijkstra's (FD)	Number of Routers
3	ZigBee-based WSN	1

$$SD = NR * aBaseSuperframeDuration * (2)^{SO} \quad (5.2)$$

$$SD = \frac{CN}{RC} * aBaseSuperframeDuration * (2)^{SO} \quad (5.3)$$

5.2.3. Message queue telemetry transport

Message queue telemetry transport (MQTT) is an IoT connectivity protocol used in the vertical domain gateway sensors to send data to cloud. Therefore, MQTT provides a lightweight method that uses publish and subscribe operations to exchange data between clients and the server. There are many types of MQTT brokers, in this study we use the open-source mosquitto [72] MQTT broker to provide communication between WSN devices and the cloud.

5.2.3.1. MQTT Communication

The MQTT mosquitto broker is responsible for sending the message from the publisher to the client subscribed via a specific topic. Figure 5.4. shows the communication between the MQTT broker and MQTT clients which these clients can be as publish or subscribe. Many MQTT receivers can subscribe to different topics, and mosquitto distributes messages from publishers according to their topics to the correct subscribers.

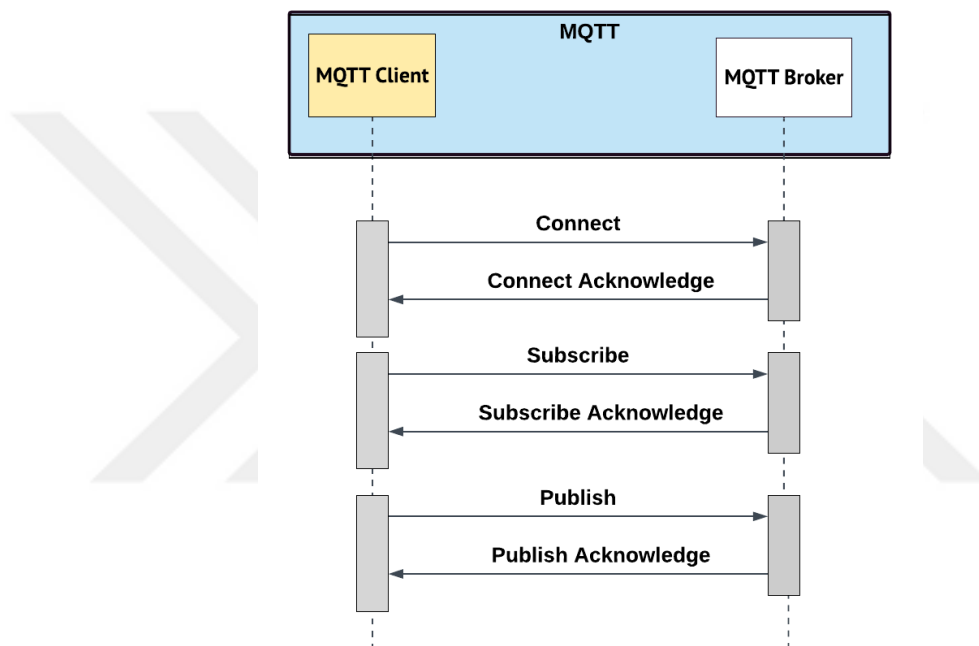


Figure 5.4. MQTT broker and client communication.

In our system, we build MQTT publish in the gateways (SDNC and destination SN), through these portals, the flow of data directed from the vertical domain to the horizontal domain in real-time. These gateways are a part of the vertical domain system and is connected to MQTT broker (mosquitto). MQTT subscribe receives the data stream from the MQTT broker to NodeJS application server. Thereafter, NodeJS sends the streaming data to the web application as shown in Figure 5.6.

5.2.4. The ESWFD system operations

The sequence diagram shown in Figure 5.5. presents the end-to-end data delivery operation of the proposed system. It clarifies the main operations performed to transfer the data and the flow table from the vertical domain to the horizontal domain. The vertical domain that is designed and simulated using the Riverbed Modeler has source node, SDN controller node, intermediate nodes and destination node, as can be seen from the Figure 5.1. and Figure 5.5. On the other hand, the horizontal domain of the proposed architecture has been built using real-world technologies such as MQTT, and NodeJS.

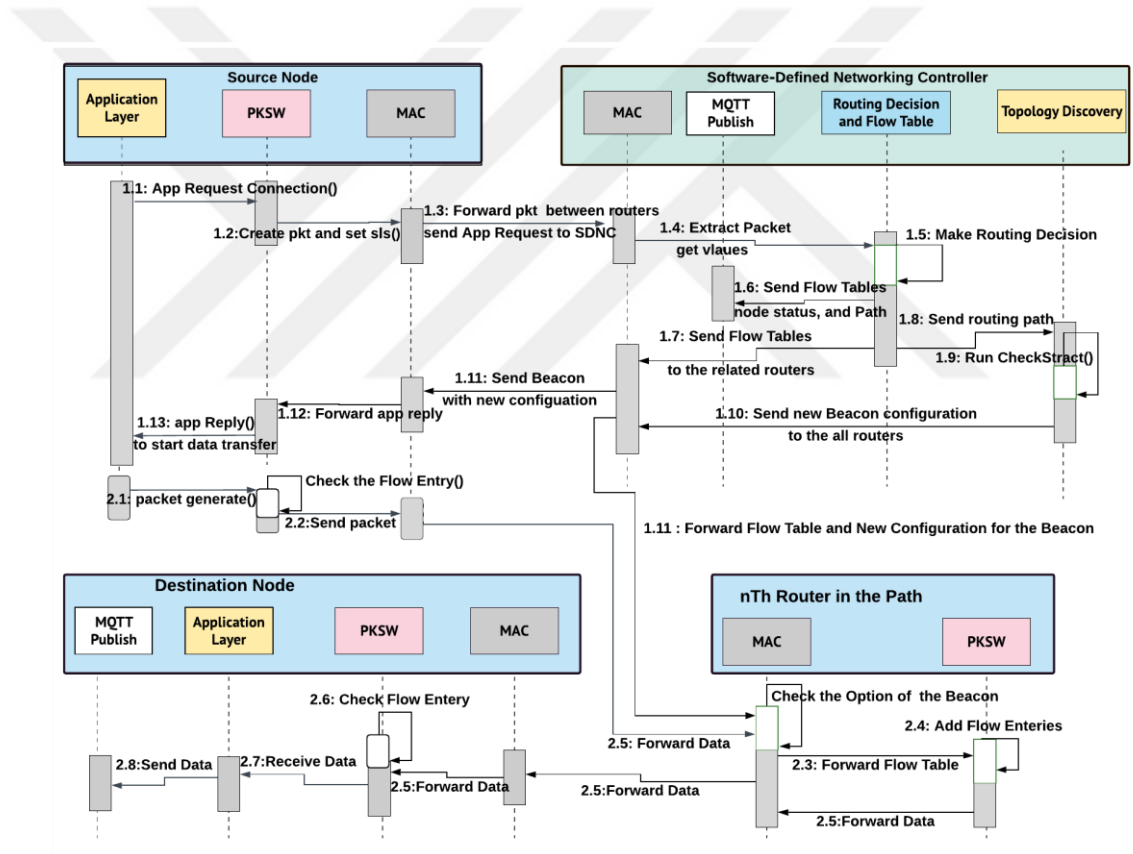


Figure 5.5. The sequence diagram for end-to-end data delivery operations.

The system operations related to connection establishment and data transfer are described in the diagram shown in Figure 5.5. After the topology discovery operation has been completed, the source node uses the Application layer module to send a connection request packet to the to the Packet Switch (PKSW) module (step 1.1-1.4).

The request is transmitted into a command frame during the contention access period until it reaches the SDNC node model. SDNC receives this connection request at the Routing Decision Flow Table (RDFT) module. RDFT then uses the Fuzzy-based Dijkstra algorithm to create the optimal path between the source and destination nodes. Through this path, it builds a flow table for the application and the relevant nodes on the path (step 1.5). Accordingly, RDFT unit sends the flow table, nodes' status information, and application path to MQTT publish module and the flow table to MAC module (steps 1.6 and 1.7) respectively. At the same time, RDFT also sends the routing path to the topology discovery to check the structure (steps 1.8-1.9). After that, topology discovery unit sends a new beacon configuration to the MAC module (step 1.10). When routers receive the beacon, they check and add the entry to its flow table and then redirect the flow table according to the next address to other nodes (steps 1.10, 2.3 and 2.4). When the source node receives a flow entry at the MAC module, the MAC unit forwards it to the PKSW module. PKSW unit inserts this entry into its flow table and then connection establishment procedure is completed (steps 1.11-1.13). At this point, the source node begins to send data packets (steps 2.1 and 2.2). Next, the PKSW module checks the address of the next hop using its flow entries to forward packets. The same procedures are repeated for each router node to forward the packets (steps 2.3-2.5). After that, the destination node receives the data, it sends this data to the MQTT publish module (steps 2.5-2.8) to be handed out to the cloud.

5.3. Experimental Results and Discussions

In Figure 5.7. the Riverbed Modeler simulation of the system was run for 1800 seconds. The source node generates data (i.e., temperature, humidity, and CO₂) in every 2 seconds. The SDN controller data are the status information of the nodes (i.e., remaining energy and neighbours of the node with related SNR), flow table and application' path. The SDN controller and the source nodes send their data to MQTT mosquitto broker using the MQTT publisher. NodeJS web interface as shown in Figure 5.6. receives these data using MQTT subscribe. arrived at NodeJS web interface and the application's data arrived at NodeJS web interface from the source node application.

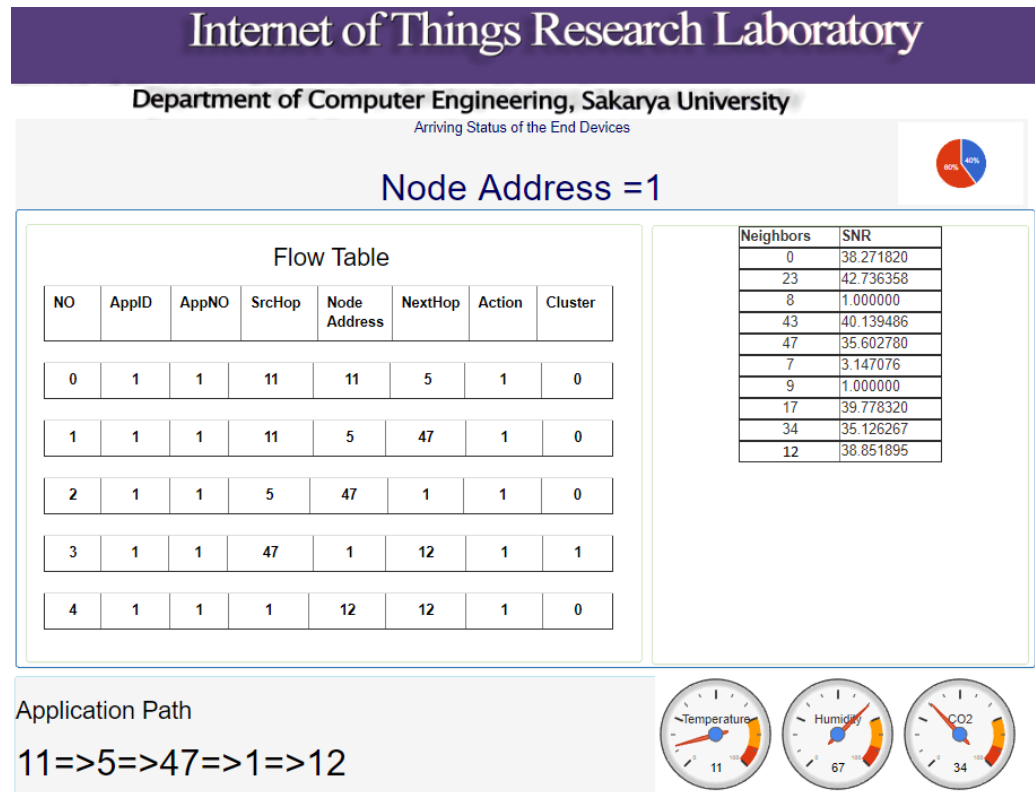


Figure 5.6. NodeJS information interface.

5.3.1. Simulation scenario and discussion

In this scenario, the simulation model in [12] was enhanced to evaluate the network's scenario as shown in Figure 5.7., which includes 50 sensor nodes and SDN controller node. The data travelled from source node to destination node via the routers in the application path that has generated from SDN controller node.

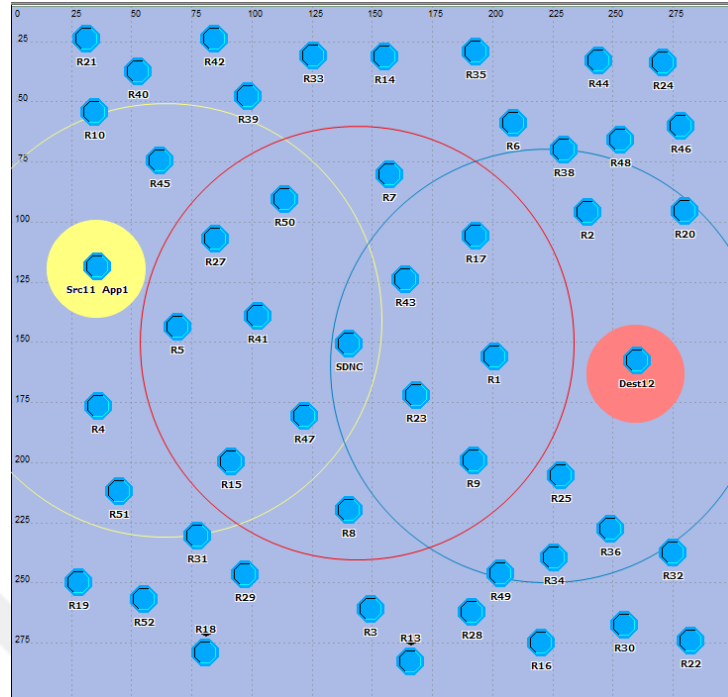


Figure 5.7. The simulation model of the SDN controller and 50 sensor nodes

5.3.2. Performance analysis and scenario results

The Simulation configuration for this scenario made for Fuzzy-based Dijkstra's (FD), Fuzzy-based Dijkstra's Beacon (FDB) and Zigbee shown in the Table 5.2. The results show that FDB performs better than FD and Zigbee in terms of Throughput, End-to-End Delay and Jitter shown in Figures (5.8, 5.9 and,5.10) respectively.

Table 5.2. Parameters of the Simulation

Group Name	Name	Value
Topology of the Network scenario	Number of end devices	50
	Network coverage area	300 m × 300 m
	SDNC location (x, y)	(150, 150)
	Simulation time	1800 s
Application configuration	Start time	50 s
	Packet interarrival time	2 s
	Packet Size	280 bits

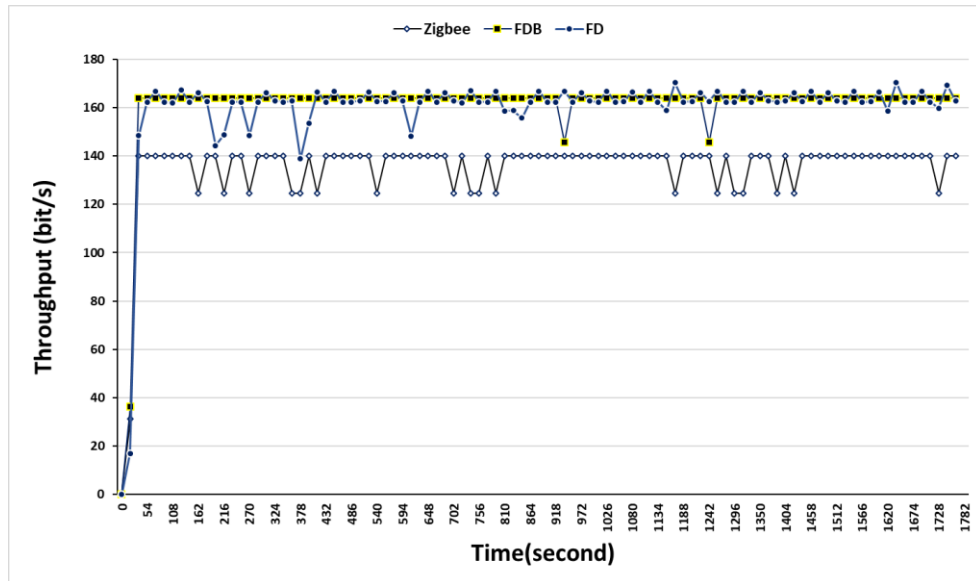


Figure 5.8. Throughput (bit/s) for 50 nodes and one application

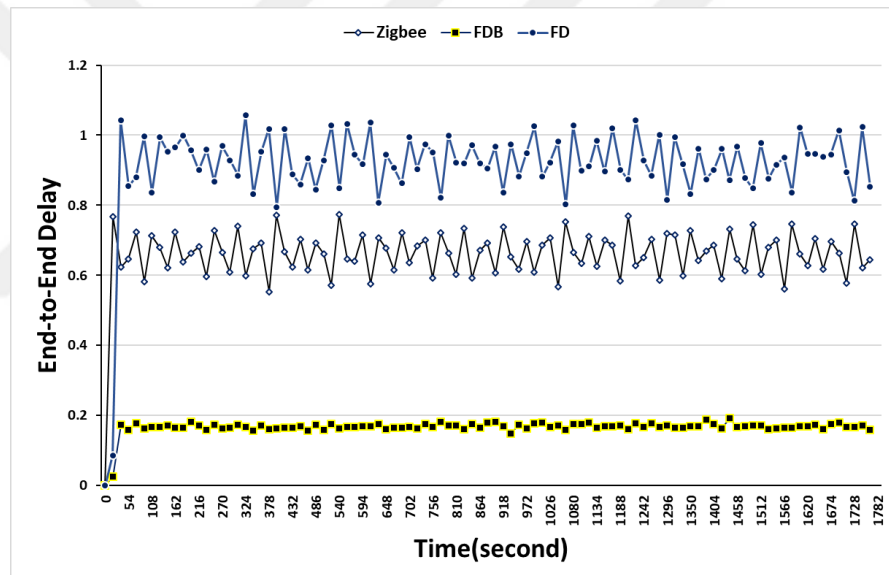


Figure 5.9. End-to-End Delay for 50 nodes and one application

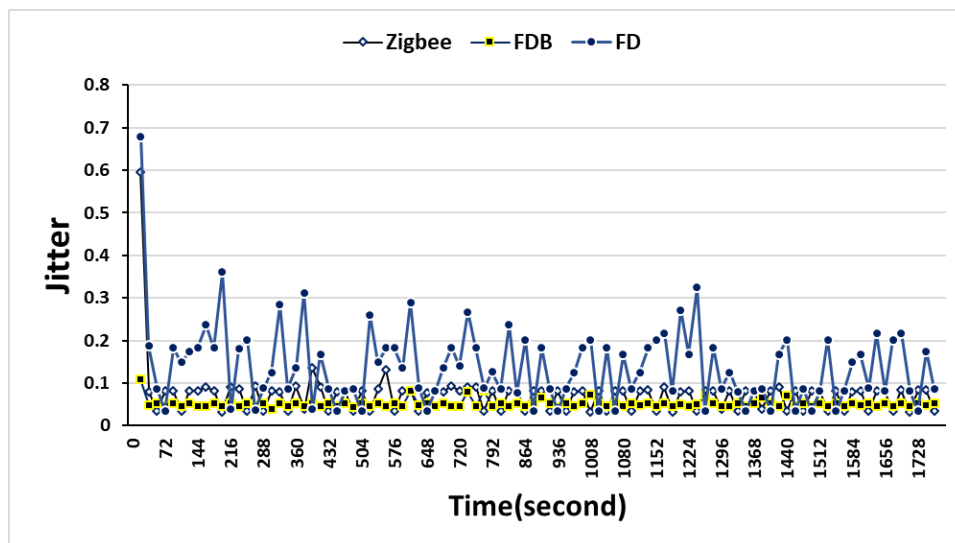


Figure 5.10. Jitter for 50 nodes and one application

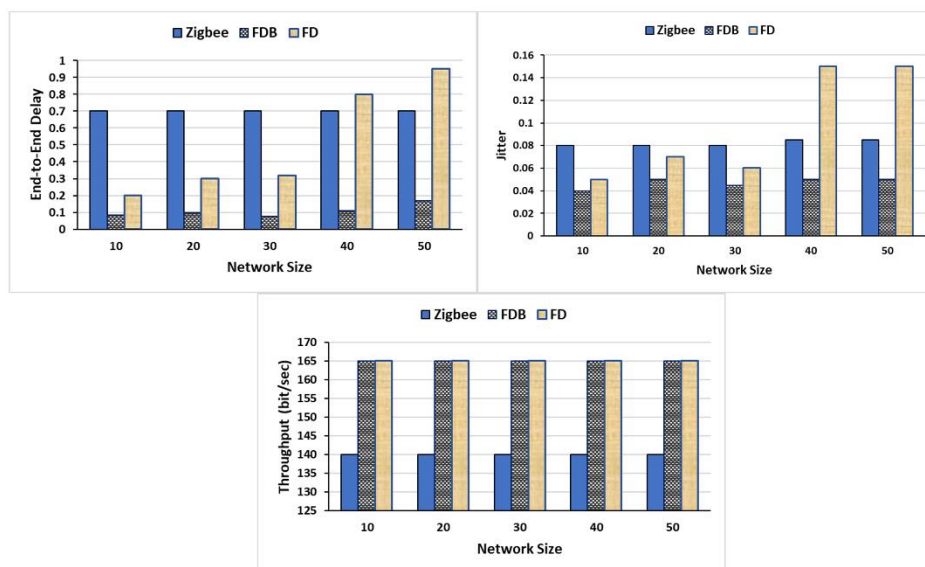


Figure 5.11. Delay, Jitter and Throughput on different network sizes

CHAPTER 6. OVERALL CONCLUSION AND FUTURE WORK

We have made a number of contributions and investigations into this thesis, where the SDN-based WSN proposed as a new structure to solve the problem statement of the thesis. We discussed the main problems of this research, namely, the development of a new energy-aware routing mechanism in the SDN controller. Secondly, the integration of the SDN-based WSN for the Internet of Things using WSN and RFID technologies. The thesis contains of six chapters. The main contributions are involved in four chapters. In this chapter, we summarize the overall conclusion for each chapter and future work.

Chapter 1 provided a motivation, problem statement, the goal of the research, research contributions, and the related work to the thesis. Moreover, the SDN-based WSN routing protocols were discussed in this chapter.

Chapter 2 discussed the integration of SDN-based WSN for Internet of Things. Also, we explained the challenges of this integration, most importantly energy routing management. The software-defined networking architecture for IoT is discussed in this chapter which contained SDN application plane, SDN control plane, SDN data plane and WSNFlow protocol stack. IEEE 802.15.4 and superframe structure are described in detail.

Chapter 3 presented the architecture design of routing protocol in the SDN controller. the SDN controller components and algorithms such as combined flow tables, network topology, node status, Dijkstra's algorithm, and fuzzy logic are discussed in this chapter. The components of the proposed system are modelled and simulated using the Riverbed Modeler software for more realistic performance evaluation.

Simulation results showed software-defined networking in the proposed system have provided a management and control solution to WSNs. In addition, the proposed system can reconfigure WSNs after deployment and is capable of both providing an effective cluster routing when finding the most efficient route and prolonging the network lifetime.

The energy consumption results showed the proposed SDN-based WSN with a fuzzy-based Dijkstra algorithm is performed better than the regular Dijkstra's and ZigBee counterparts. Furthermore, the SDN appeared as the most powerful candidate to solve the problem of network flexibility in WSN among deployment options.

The performance metrics are the power consumption ratio and SNR used in the proposed system when the application path is established between WSN devices. Therefore, in future work, the proposed system needs to increase the number of performance metrics according to the needs of the IoT application.

Chapter 4 presented a new real-time IoT data analytics architecture for healthcare architecture. The proposed system composed of an SDN-enabled WSN and RFID in the vertical domain and of data analytics tools such as Kafka, Spark, MongoDB and NodeJS in the horizontal domain. The developed system has been tested for real-time detection of Wolf Parkinson White syndrome using logistic regression method. The obtained results on the Arrhythmia dataset revealed that this disease can be predicted with high accuracy (i.e. %70 in average) using the proposed architecture, in real-time. Considering the fact that the proposed system includes high performance and scalable data analytics technologies such as Kafka and Spark, it is not difficult to conclude that this new architecture can be used effectively in real-time big data processing applications. To model and simulate any IoT enabling technology, the developed system also communicates with Riverbed Modeler using TCP sockets. So, the proposed architecture can be used as a time-saving experimental environment for any IoT-based system.

In addition, to integrate IoT technology – namely radio frequency identification into the WSN source devices as part of vertical domain devices. Furthermore, the communication and implementation between the vertical and horizontal domain devices will be included in the chapter.

Therefore, an effective data routing strategy with low power consumption for WSNs should be smart and flexible in the SDN controller to meet IoT requirements.

Chapter 5 presented an enhanced SDN-based WSN fuzzy-based Dijkstra, ESWFD, which can reconfigure the SDNC's beacon after the application path is established. ESWFD is also designed with an IoT gateway between ESWFD and MQTT to support IoT applications in real-time. The IoT gateway is used to collect data in order to send it to the cloud. The IoT cloud requires a network to support real-time data transfer in order to analyse the data for some IoT applications.

The ESWFD improves the system performance, which was compared with a traditional ZigBee, and the Fuzzy-based Dijkstra. The results increased the capacity of network throughput and reduced packet delay and jitter.

The improvement in delay results is required for some applications, where each application requires different QoS support. For example, delay is another important measure of performance, especially for time-critical services. For future work need to evaluate the QoS requirements for applications and to be able to manage the network using a web interface.

REFERENCES

- [1] C. Buratti *et al.*, “Testing Protocols for the Internet of Things on the EuWIn Platform,” *IEEE Internet Things J.*, vol. 3, no. 1, pp. 124–133, Feb. 2016.
- [2] D. Evans, “The Internet of Things How the Next Evolution of the Internet Is Changing Everything,” *CISCO White Pap.*, no. April, pp. 1–11, 2011.
- [3] L. Mainetti, L. Patrono, and A. Vilei, “Evolution of wireless sensor networks towards the Internet of Things: A survey,” *SoftCOM*, pp. 1–6, 2011.
- [4] A. B. Al-Shaikhli, C. Çeken, and M. Al-Hubaishi, “WSANFlow: An Interface Protocol Between SDN Controller and End Devices for SDN-Oriented WSAN,” *Wirel. Pers. Commun.*, vol. 101, no. 2, pp. 755–773, 2018.
- [5] A. W. Burange and H. D. Misalkar, “Review of Internet of Things in development of smart cities with data management & privacy,” in *2015 International Conference on Advances in Computer Engineering and Applications*, 2015, pp. 189–195.
- [6] R. Gorrepotu, N. S. Korivi, K. Chandu, and S. Deb, “Sub-1GHz miniature wireless sensor node for IoT applications,” *Internet of Things*, vol. 1–2, pp. 27–39, 2018.
- [7] E. G. M. Petrakis, S. Sotiriadis, T. Soultanopoulos, P. T. Renta, R. Buyya, and N. Bessis, “Internet of Things as a Service (iTaaS): Challenges and Solutions for Management of Sensor Data on the Cloud and the Fog,” *Internet of Things*, vol. 3–4, pp. 156–174, 2018.
- [8] M. M. Warriar and A. Kumar, “An Energy Efficient Approach for Routing in Wireless Sensor Networks,” *Procedia Technol.*, vol. 25, no. Raerest, pp. 520–527, 2016.
- [9] R. E. Mohamed, A. I. Saleh, M. Abdelrazzak, and A. S. Samra, “Survey on Wireless Sensor Network Applications and Energy Efficient Routing Protocols,” *Wirel. Pers. Commun.*, vol. 101, no. 2, pp. 1019–1055, 2018.
- [10] W. Xiang, N. Wang, and Y. Zhou, “An Energy-Efficient Routing Algorithm for Software-Defined Wireless Sensor Networks,” *IEEE Sens. J.*, vol. 16, no. 20, pp. 7393–7400, 2016.

- [11] Y. Wang, H. Chen, X. Wu, and L. Shu, "An energy-efficient SDN based sleep scheduling algorithm for WSNs," *J. Netw. Comput. Appl.*, vol. 59, pp. 39–45, 2016.
- [12] M. Al-Hubaishi, C. Çeken, and A. Al-Shaikhli, "A novel energy-aware routing mechanism for SDN-enabled WSN," *Int. J. Commun. Syst.*, p. e3724, Jun. 2018.
- [13] A. Sarkar and T. Senthil Murugan, "Routing protocols for wireless sensor networks: What the literature says?," *Alexandria Eng. J.*, vol. 55, no. 4, pp. 3173–3183, 2016.
- [14] J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: A survey," *IEEE Wirel. Commun.*, vol. 11, no. 6, pp. 6–27, 2004.
- [15] D. P. Bhoomika and D. P. Ashish, "Hierarchical Routing Protocols in Wireless Sensor," *Int. Journal Comput. Technol. Appl. an open access Int. J. peer Rev. online J.*, vol. 6, no. 1, pp. 47–52, 2012.
- [16] S. Saranya and M. Princy, "Routing techniques in sensor network -a survey," *Procedia Eng.*, vol. 38, pp. 2739–2747, 2012.
- [17] A. S. Toor and A. K. Jain, "A survey of routing protocols in Wireless Sensor Networks: Hierarchical routing," *2016 Int. Conf. Recent Adv. Innov. Eng. ICRAIE 2016*, pp. 1–6, 2017.
- [18] S. P. Singh and S. C. Sharma, "A survey on cluster based routing protocols in wireless sensor networks," *Procedia Comput. Sci.*, vol. 45, no. C, pp. 687–695, 2015.
- [19] M. Sood, "Software defined network - Architectures," in *2014 International Conference on Parallel, Distributed and Grid Computing*, 2014, pp. 451–456.
- [20] M. Karakus and A. Durresi, "A scalable inter-AS QoS routing architecture in software defined network (SDN)," in *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, 2015, vol. 2015-April, pp. 148–154.
- [21] R. Pradeepa and M. Pushpalatha, "SDN Enabled SPIN Routing Protocol for Wireless Sensor Networks," *2016 Int. Conf. Wirel. Commun. Signal Process. Netw.*, vol. 10, no. 6, pp. 639–643, 2016.
- [22] R. C. A. Alves, D. A. G. Oliveira, G. C. C. F. Pereira, B. C. Albertini, and C. B. Margi, "WS 3 N : Wireless Secure SDN-Based Communication for Sensor Networks," *Secur. Commun. Networks*, vol. 2018, pp. 1–14, Aug. 2018.

- [23] A. Hakiri and A. Gokhale, "Rethinking the design of LR-WPAN IoT systems with software-defined networking," in *Proceedings - 12th Annual International Conference on Distributed Computing in Sensor Systems, DCOSS 2016*, 2016, pp. 238–243.
- [24] L. F. D. S. Santos, F. F. De Mendonca, and K. L. Dias, "SDN: An SDN-Based Routing Architecture for Wireless Sensor Networks," *Brazilian Symp. Comput. Syst. Eng. SBESC*, vol. 2017-Novem, pp. 63–70, 2017.
- [25] N. Kumar and D. P. Vidyarthi, "A Green Routing Algorithm for IoT-Enabled Software Defined Wireless Sensor Network," *IEEE Sens. J.*, vol. 18, no. 22, pp. 9449–9460, 2018.
- [26] K. Sood, S. Yu, and Y. Xiang, "Software-Defined Wireless Networking Opportunities and Challenges for Internet-of-Things: A Review," *IEEE Internet Things J.*, vol. 3, no. 4, pp. 453–463, 2016.
- [27] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A software defined networking architecture for the internet-of-things," *IEEE/IFIP NOMS 2014 - IEEE/IFIP Netw. Oper. Manag. Symp. Manag. a Softw. Defin. World*, 2014.
- [28] A. De Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," *2014 27th Bienn. Symp. Commun.*, pp. 71–75, 2014.
- [29] Z. Kerravala, "The Top Five Network Problems Solved by SDNs," 2019. [Online]. Available: <https://blog.silver-peak.com/the-top-five-network-problems-solved-by-sdn>. [Accessed: 29-Jul-2019].
- [30] I. Journal, "Internet of Things : A Survey of IoT Applications based on their desirable device characteristics," *Int. J. Recent Eng. Res. Dev.*, vol. 02, no. 10, pp. 10–20, 2017.
- [31] G. Sinnapolu and S. Alawneh, "Integrating wearables with cloud-based communication for health monitoring and emergency assistance," *Internet of Things*, vol. 1–2, pp. 40–54, 2018.
- [32] W. Stallings, *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*, vol. 4301, no. 3. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.
- [33] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A Software Defined Networking architecture for the Internet-of-Things," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014, no. 5-9 May 2014, pp. 1–9.

- [34] L. Hu, J. Wang, E. Song, A. Ksentini, A. Hossain, and M. Rawashdeh, "SDN-SPS : Semi-Physical Simulation for Software-Defined Networks," vol. 16, no. 20, pp. 7355–7363, 2016.
- [35] B. Trevizan de Oliveira, R. Cerqueira Afonso Alves, and C. Borges Margi Universidade de São Paulo São Paulo, "Software-Defined Wireless Sensor Networks and Internet of Things Standardization Synergism," *2015 IEEE Conf. Stand. Commun. Netw.*, pp. 60–65, 2015.
- [36] P. Jayashree and F. Infant Princy, "Leveraging SDN to conserve energy in WSN-An analysis," *2015 3rd Int. Conf. Signal Process. Commun. Netw.*, 2015.
- [37] D. Sinh, L. Le, B. P. Lin, and L. Tung, "SDN / NFV - A new approach of deploying network infrastructure for IoT," *2018 27th Wirel. Opt. Commun. Conf.*, pp. 1–5, 2018.
- [38] T. Minh, C. Nguyen, and D. B. Hoang, "S-MANAGE Protocol For Software-Defined IoT," *2018 28th Int. Telecommun. Networks Appl. Conf.*, pp. 1–6, 2018.
- [39] S. Bera, G. S. Member, S. Misra, and S. Member, "Soft-WSN : Software-Defined WSN Management System for IoT Applications," *IEEE Syst. J.*, vol. 12, no. 3, pp. 2074–2081, 2018.
- [40] F. F. J. Lasso, K. Clarke, and A. Nirmalathas, "A software-defined networking framework for IoT based on 6LoWPAN," in *Wireless Telecommunications Symposium*, 2018, vol. 2018-April, pp. 1–7.
- [41] X.-Y. Chen and Z.-G. Jin, "Research on Key Technology and Applications for Internet of Things," *Phys. Procedia*, vol. 33, pp. 561–566, 2012.
- [42] S. Tennina *et al.*, *IEEE 802.15.4 and ZigBee as Enabling Technologies for Low-Power Wireless Systems with Quality-of-Service Constraints*, vol. 1. 2013.
- [43] T. Ok, S. Baek, and B. Dae, "Performance analysis of IEEE 802 . 15 . 4 superframe structure with the inactive period," *Perform. Eval.*, vol. 106, pp. 50–69, 2016.
- [44] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," *21st Conf. IEEE Comput. Commun. Soc.*, vol. 3, no. 3, pp. 1567–1576, 2002.
- [45] W. Braun and M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," pp. 302–336, 2014.
- [46] A. S. Yuan, H.-T. Fang, and Q. Wu, "OpenFlow Based Hybrid Routing in Wireless Sensor Networks," *Intell. Sensors, Sens. Networks Inf. Process. (ISSNIP), 2014 IEEE Ninth Int. Conf.*, vol. 1, no. April, pp. 21–24, 2014.

- [47] Y. Dasgupta and P. M. G. Darshan, “Application of Wireless Sensor Network in remote monitoring: Water-level sensing and temperature sensing, and their application in agriculture,” *1st Int. Conf. Autom. Control. Energy Syst. - 2014, ACES 2014*, 2014.
- [48] L. Hu, J. Wang, E. Song, A. Ksentini, M. A. Hossain, and M. Rawashdeh, “SDN-SPS: Semi-Physical Simulation for Software-Defined Networks,” *IEEE Sens. J.*, vol. 16, no. 20, pp. 7355–7363, 2016.
- [49] A. Koubâa, A. Cunha, M. Alves, and E. Tovar, “TDBS: A time division beacon scheduling mechanism for ZigBee cluster-tree wireless sensor networks,” *Real-Time Syst.*, vol. 40, no. 3, pp. 321–354, 2008.
- [50] A. Lotfizadeh, “Fuzzy sets,” *Inf. Control*, vol. 8, no. 3, pp. 338–353, 1965.
- [51] J. Mocq, A. St-Hilaire, and R. A. Cunjak, “Assessment of Atlantic salmon (*Salmo salar*) habitat quality and its uncertainty using a multiple-expert fuzzy model applied to the Romaine River (Canada),” *Ecol. Modell.*, vol. 265, no. September, pp. 14–25, 2013.
- [52] E. W. Dijkstra, “A Note on Two Problems in Connexion with Graphs,” vol. 271, no. 1, pp. 269–271, 1959.
- [53] L. Mainetti, L. Palano, L. Patrono, M. L. Stefanizzi, and R. Vergallo, “Integration of RFID and WSN technologies in a Smart Parking System,” *2014 22nd Int. Conf. Software, Telecommun. Comput. Networks, SoftCOM 2014*, pp. 104–110, 2014.
- [54] T. Adame, A. Bel, A. Carreras, J. Melià-Seguí, M. Oliver, and R. Pous, “CUIDATS: An RFID–WSN hybrid monitoring system for smart health care environments,” *Futur. Gener. Comput. Syst.*, vol. 78, pp. 602–615, 2018.
- [55] S. Mirshahi and S. Uysal, “Integration of RFID and WSN for Supply Chain Intelligence System,” in *Electronics, Computers and Artificial Intelligence (ECAI), 2013 International Conference on*, 2013, no. 27-29 June 2013, pp. 1–6.
- [56] S. R. Vijayalakshmi and S. Muruganand, “A survey of Internet of Things in fire detection and fire industries,” *Proc. Int. Conf. IoT Soc. Mobile, Anal. Cloud, I-SMAC 2017*, pp. 703–707, 2017.
- [57] W. Liang, S. Xie, J. Long, K.-C. Li, D. Zhang, and K. Li, “A Double PUF-based RFID Identity Authentication Protocol in Service-Centric Internet of Things Environments,” *Inf. Sci. (Ny)*, vol. 503, pp. 129–147, 2019.
- [58] J. V. V. Sobral *et al.*, “A framework for enhancing the performance of Internet of Things applications based on RFID and WSNs,” *J. Netw. Comput. Appl.*, vol. 107, no. January, pp. 56–68, 2018.

- [59] F. J. Valente and A. C. Neto, "Intelligent steel inventory tracking with IoT / RFID," *2017 IEEE Int. Conf. RFID Technol. Appl. RFID-TA 2017*, pp. 158–163, 2017.
- [60] C. P. Tang, Z. Y. Tang, Y. H. Yang, and Y. J. Zhan, "WSID identification platform of heterogeneous networks based on RFID and WSN," *Proc. 2010 IEEE Int. Conf. RFID-Technology Appl. RFID-TA 2010*, no. June, pp. 217–221, 2010.
- [61] O. Abdulkader, A. M. Bamhdi, V. Thayananthan, K. Jambi, and M. Alrasheedi, "A novel and secure smart parking management system (SPMS) based on integration of WSN, RFID, and IoT," *2018 15th Learn. Technol. Conf. L T 2018*, pp. 102–106, 2018.
- [62] F. Aktas, C. Ceken, and Y. E. Erdemli, "IoT-Based Healthcare Framework for Biomedical Applications," *J. Med. Biol. Eng.*, pp. 1–14, Dec. 2017.
- [63] N. Garg, *Apache Kafka*. 2014.
- [64] Apache Spark, "Apache Spark™ - Unified Analytics Engine for Big Data," *spark.apache.org*, 2018. [Online]. Available: <https://spark.apache.org/>. [Accessed: 21-Oct-2018].
- [65] L. Ganz, "Electrocardiography," *Goldman's Cecil Med. Twenty Fourth Ed.*, vol. 1, pp. 272–278, 2011.
- [66] M. G. Kaya, I. Ozdogru, M. Yarlioglues, T. Inanc, A. Dogan, and N. K. Eryol, "Coronary ischemia induced Wolf Parkinson White syndrome," *Int. J. Cardiol.*, vol. 129, no. 1, pp. 3–4, 2008.
- [67] B. H. Williams, "UCI Machine Learning Repository: Character Trajectories Data Set," 2008. [Online]. Available: <https://archive.ics.uci.edu/ml/datasets/arrhythmia>. [Accessed: 26-Oct-2018].
- [68] L. Hou, S. Zhao, X. Li, P. Chatzimisios, and K. Zheng, "Design and implementation of application programming interface for Internet of things cloud," *Int. J. Netw. Manag.*, vol. 27, no. 3, pp. 1–15, 2017.
- [69] P. C. Prabhu Kumar and G. Geetha, "Web-cloud architecture levels and optimized MQTT and COAP protocol suites for web of things," *Concurr. Comput.*, no. April, pp. 1–8, 2018.
- [70] P. Dhar and P. Gupta, "Intelligent parking Cloud services based on IoT using MQTT protocol," *Int. Conf. Autom. Control Dyn. Optim. Tech. ICACDOT 2016*, pp. 30–34, 2017.

- [71] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, “MQTT-S - A publish/subscribe protocol for wireless sensor networks,” *3rd IEEE/Create-Net Int. Conf. Commun. Syst. Softw. Middleware, COMSWARE*, pp. 791–798, 2007.
- [72] R. A Light, “Mosquitto: server and client implementation of the MQTT protocol,” *J. Open Source Softw.*, vol. 2, no. 13, p. 265, May 2017.
- [73] S. Tilkov and S. Vinoski, “Node.js: Using JavaScript to Build High-Performance Network Programs,” *IEEE Internet Comput.*, vol. 14, no. 6, pp. 80–83, Nov. 2010.
- [74] S. Zhang, X. Chen, and S. Wang, “Research on the monitoring system of wheat diseases, pests and weeds based on IOT,” in *2014 9th International Conference on Computer Science & Education*, 2014, no. Iccse 22-24 August, pp. 981–985.
- [75] O. León, J. Hernández-Serrano, and M. Soriano, “An efficient energy-aware predictive clustering approach for vehicular ad hoc networks Rasmeet,” *Int. J. Commun. Syst.*, vol. 23, no. 5, pp. 633–652, 2015.

RESUME

Mohammed Al-Hubaishi is a lecturer and researcher at the faculty of computer science and information system, Tamar University, Yemen. He received his B.Sc. degree in computer science from Tamar University, Yemen, in 2002. In December 2010, he has M.Sc. degree in wireless ad hoc network from department of electronic and informatics engineering faculty of university of Algarve, Portugal. His research interests include wireless network, ad hoc network, routing protocols, Fuzzy logic, SDN, IoT, WSN, Kafka, Spark, NodeJS, machine learning and Riverbed simulation. He is currently pursuing his Ph.D. degree in computer and information engineering at Sakarya University. He is expected to be graduated in 2019.