Design and Implementation of IoT Based Smart Greenhouse Monitoring System

by

Jyoti Raj Sharma Subedi

A thesis submitted
In partial fulfillment of the requirements
For the Master of Applied Science in Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Internet of Things (IoT) has drawn much attention in recent years. With IoT, physical world entities get connected through internet. IoT is used currently in various applications, such as environmental monitoring, control systems, farming, home automation, security and surveillance systems etc.

The aim of this research is to design a low-cost, energy-efficient, reliable and scalable embedded application for the smart greenhouse monitoring system. The IoT based system designed in this thesis uses various sensors to measure the air and soil quality parameters in the greenhouse, and monitor real-time data online using web-server and mobile phone based applications. A ZigBee based wireless sensor network is implemented to transport various sensory data to the gateway. Among other contributions, the designed system develops a new routing algorithm by introducing a confirmed delivery of packets and re-routing features. We also introduced an efficient cost metric for making routing decisions within WSN using hops count, and simple bi-directional link quality estimator using PRR and current battery voltage of neighbor nodes. We also verified the stability of the system by conducting various performance tests. The system is equipped with data analytic functions for the online examination of the data. The designed system adopts event-based triggering and data aggregation methods to reduce the number of transmissions, and develops a new algorithm for such purpose. The web-server and mobile applications have user interface to display the output of the data analytic services, warning, control operations and give access to query data of the user's interest.

Acknowledgements

I would like to express my humble gratitude to my supervisor Professor Dimitrios Makrakis. His support and motivation throughout my academic career enabled to complete this thesis in time. I would like to take this opportunity to thank him for reviewing my work regularly and providing valuable suggestions throughout this thesis.

I am thankful to all members of Broadband and Inter-networking laboratory for providing me adequate laboratory equipment. I am thankful to all faculty members of Electrical and Computer Engineering, University of Ottawa who directly or indirectly helped to complete this thesis.

Last but not the least, I would like to express gratitude to my friends and family members who motivated me and provided constant support to complete this thesis.

Contents

List of Figures

List of Tables

Abbreviations

MQTT	**Message Queue Telemetry Transport**
FG	**Functional Group**
CoAP	**Constrained Application Protocol**
HTTP	**Hyper-Text Transfer Protocol**
AWS	**Amazon Web Services**
WSN	**Wireless Sensor Networks**
IoT	**Internet of Things**
PHY	**Physical**
MAC	**Medium Access Control**
SoC	**System-on-Chip**
WDT	**Watchdog Timer**
AC	**Arithmetic Coding**
AES	**Advanced Encryption Standard**
TLS	**Transport Layer Security**
DTLS	**Datagram Transport Layer Security**
REST	**Representational State Transfer**
JSON	**JavaScript Object Notation**
GBR	**Gradient Based Routing**
LEACH	**Low Energy Adaptive Clustering Hierarchy**
APTEEN	**Adaptive Threshold sensitive Energy Efficient sensor Network**
PRR	**Packet Reception Ratio**
RF	**Radio Frequency**
IEEE	**Institute of Electrical and Electronic Engineers**
ACK	**Acknowledgement Message**
ADV	**Advertisement Message**

GHz	**Gigahertz**
MHz	**Megahertz**
HEAR	**Hierarchical Energy Aware Routing**
HAR	**Hierarchy-based Anycast Routing**
3G	**3rd Generation**
GSM	**Global System for Mobile Communication**
BLE	**Bluetooth Low Energy**
GPRS	**General Packet Radio Service**
WLAN	**Wireless Local Area Network**
WPAN	**Wireless Personal Area Network**
SPIN	**Sensor Protocols for Information via Negotiation**
TTL	**Time To Live**
MCFA	**Minimum Cost Forwarding Algorithm**
LQE	**Link Quality Estimator**
REQ	**Request Message**
PRR	**Packet Reception Ratio**
LED	**Light Emitting Diode**
QoS	**Quality of Service**
ACK	**Acknowledgment**
RSSI	**Received Signal Strength Indicator**
TX	**Transmitter**
RX	**Receiver**
AC	**Alternating Current**
DC	**Direct Current**
NOOBS	**New Out of Box Software**
USB	**Universal Serial Bus**
CPU	**Central Processing Unit**
IDE	**Integrated Development Environment**
VNC	**Virtual Network Computing**
PAN	**Personal Area Network**
HDMI	**High Definition Multimedia Interface**
NW	**Network**
JV	**Join Verification**

JN	**Join Notification**
DH	**Destination Address High**
DL	**Destination Address Low**
SQL	**Structured Query Language**
HTML	**Hyper Text Markup Language**
AJAX	**Asynchronous JavaScript And XML**
SFD	**Start of Frame**
FCS	**Frame Control Sequence**
SFD	**Start of Frame Data**
TCP	**Transmission Control Protocol**
UDP	**User Datagram Protocol**
OS	**Operating System**
ETX	**Expected Number of Transmissions**
WMEWMA	**Window Mean with Exponentially Weighted Moving Average**
RNP	**Required Number of Packet Reception**
ARR	**Acquired Reception Ratio**

Chapter 1

Introduction

Sensors are devices which sense the physical environment and provide measurements of various physical parameters in the form of electrical signals. Sensors measuring physical environmental parameters are readily available in the market, and are becoming more accurate, smaller, cheaper, making them suitable and easily usable in embedded system designs. They can be used for environmental monitoring, home-automation, military applications, crops health monitoring/precision farming etc. Online monitoring and automation were not available up until relatively recently. For example, air and soil quality monitoring was implemented by collecting physically water and soil samples, and sending them to relevant laboratories (away from the location of farming) for analysis. The time duration elapsing between collection of the sample to getting the results was usually quite lengthy. The growing accuracy of modern sensors, combined with the convenient transportation of information through wireless networks and the expansive growth of online services, provided the foundation for the development of a series of applications that revolutionized farming, by simplifying considerably the gathering of samples, reducing considerable the time between collection and processing of those samples, and cutting down the cost the cultivator has to face in order to have those results made available. These new applications fall under the terms of intelligent farming and precision farming. At present, a new transformation of telecommunications and information technologies infrastructure is happening, as we move from Internet focused network to an Internet of Things (IoT) focus networking. Incorporation of IoT generated new applications generates the potential for new considerably more advanced, time and cost cutting precision farming.

1.1 Precision Farming System

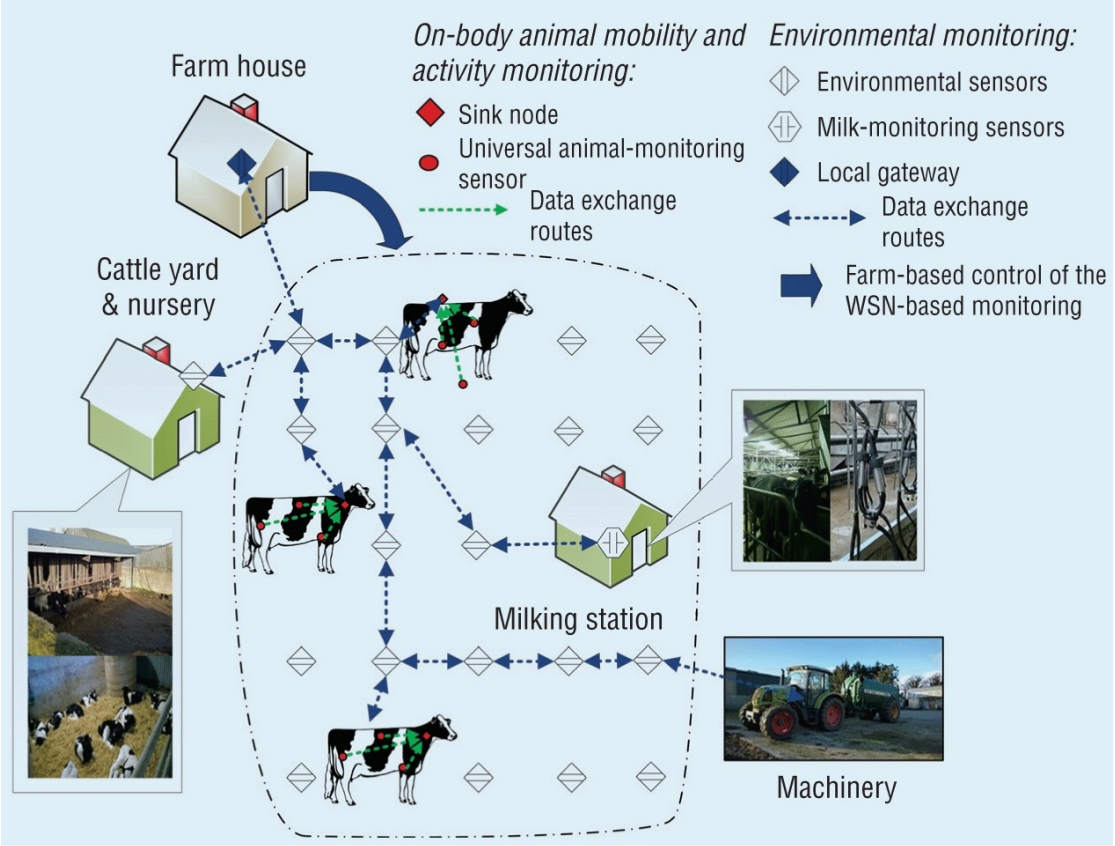

FIGURE 1.1: Description of modern precision farming system [1]

Farming is very critical to human race, as lack of adequate food production has been proved to be source of famine, war, and social instability. The systematic farming approach includes cultivating and harvesting crops under the direct or indirect supervision of farmers using advanced technologies. Undoubtedly, farming is the life-line for many rural areas and has gradually become more prevalent in urban areas including cities where we experience vertical farming [2]. Modern advanced methodologies have appeared in support of intelligent farming, also known as precision farming, for the production of high quality crops at high volume. A precision farming system includes monitoring, immediate notification and response when crops experience harmful changes, automation and control etc.

The needs of wireless sensor networks for precision farming are obvious, such as the needs of remote/online monitoring, modern farming paradigm shift, distant diagnosis, and asset tracking, as depicted in Figure 1.1. Use of wireless sensor networks (WSN)

technologies can significantly benefit various processes occurring during the farming cycle, starting from cultivation and all the way to harvesting of crops. Wireless sensor networks enable collection, storage and exchange of sensor data. A precision farming system making use of IoT and WSN plays a vital role in precision farming by improving the monitoring and control processes. Various aspects of farming, for example, irrigation, harvesting, measuring quality of air and soil, measuring vegetation indices, monitoring of temporal stability, distributed valley irrigation, use of alarming system, can benefit by the combine use of IoT and WSN technologies.

In farming sectors, air quality parameters, for example, temperature, heat-index, humidity, dew-point, carbon-dioxide (CO2), carbon-monoxide (CO), methane (CH4), light intensity level and soil quality indices (such as soil moisture, PH level) affect directly or indirectly the health of the crops [2]. The air quality parameters remain fluctuating because of seasonal or sudden erratic weather changes and pollutants content in the environment. Artificial environment created for farming, for example, greenhouse sectors, contain various electronics devices, such as light emitting diodes, which produce fluctuation to the temperature levels of the greenhouse. Also, the closed in-house air contains pollutants and unwanted gases which tend to accumulate. The situation becomes problematic when manual monitoring and provisioning is used.

Temperature is a crucial factor to the growth and development of the crops and regulates the rate at which physiological process occur. Temperature must be kept in certain range for any vital process to be sustained and enable grow. The response to temperature is conditioned by other environmental parameters as well, e.g. humidity, moisture. According to [2] and several of the references provided in it, the optimum temperature range for the growth of most crops cultivated in a greenhouse is between 20 to 25 °C. For temperatures below 10 °C, the growth of most crops drops drastically but some growth occurs and keeps healthy plants temperatures as low as 5 °C. Temperatures above 25 °C is not preferred as it affects negatively the plants' growth. Temperatures above 35 °C may even completely stop the growth of the plants, even with sufficient moisture present. Photosynthesis occurring at the leaves of the plants is blocked by high-temperature [2]. Photosynthesis can be blocked due to damage occurring on the plant's leaves' tissues due to high temperature.

Relative humidity keeps fluctuating with change in temperature and the transpiration

process increases the moisture content of the greenhouse air. Maintainability of the relative humidity below a threshold value is a common practice and is achieved by regulating amount of water content on the basis of if t is day or night. Generally, it is wished that the humidity is kept at low level.

Water is crucial for all living beings. So, plants require adequate water supply to grow properly. Lack of adequate water directly affects the growth, and survival ability of the plants; it could even initiate their decay immediately. Deficiency of water in plant tissues occurs when the rate of transpiration exceeds that of water consumption by the roots [3]. Lack of adequate water in soil decreases optimal photosynthesis and slows the growth of roots [3].

Transpiration is the process through which plants carry water content from roots to the small pores on their leaves, and from there slowly evaporates to the atmosphere. According to [3], plants contribute an additional 10-15 % of water content in the atmosphere via this process and transpire water that weights many times the plant's weight during its development period.

In the photosynthesis of the plants, humidity and temperature play a crucial role. Photosynthesis is the process through which plants generate energy for their growth, and is based on CO2 absorption, light exposure and water content in the plant. The quality of the plant in terms of growth and development is directly proportional to how well the photosynthesis process is performed.

Light is a critical factor to the growth and the development of the plants. However, different plants have different light exposure requirements. Both, inadequate and excessive light intensities are injurious. According to [4], a plant's optimal exposure to light increases the rate of photosynthesis. Lack of light exposure results in the plants developing unusually extended inter-nodes, generates "whiteness" on the leaves and stops the growth of the roots. Very high light intensity should also be avoided as it can burn the leaves and reduces the crop production.

In summary, careful management is required in terms of monitoring soil and air quality parameters; a task that is assigned to precision farming system.

1.2 Motivation of Research

As of today, a large number of farming enterprises, whether they are large-scale (corporate) or small-scale (e.g. family run) still are not fully emerged in the use of advanced precision farming technologies. Farmers are still dependant on having tests mount off the premises (in various far from the farm's location laboratories), with the results becoming available after considerable lapse of time. It is accurate to say that use of modern information and communication services, including the emerging IoT have not been adopted by the farming industry nor used aggressively in their infrastructure and operational process. This realization directed us to the conclusion that, should we design and develop a plants health monitoring system that combines state-of-the-art IoT technologies and precision farming, we would have made a valuable contribution to the farming sector, probably at global scale.

1.3 Scope of Research

Scope of this research is the thorough design and implementation of an intelligent real-time plants health monitoring system that makes uses of advanced sensing, WSN and IoT technologies. The designed system is suitable for greenhouse farming. A closed loop monitoring system is developed, which acquires sensor data, processes and transforms it to visual information that is easily understood and made use of by the growers. The visually displayed information is accessible through web-server access and Android based mobile applications. A ZigBee based wireless sensor network is designed for the transportation of sensor collected data. It provides a very flexible, scalable, and easily re-configurable data transport solution, which is as much suitable for large cultivation areas, as is for small agricultural businesses. The designed system enables the cultivator to identify the plant diseases at its initial stages. It does so by analyzing sensor data from past samples using descriptive data analytic services.

1.4 Objectives of Research

The research objective is to contribute to the advancement of precision farming technologies by designing a crops monitoring system that makes use of WSN and IoT technologies. The developed system has the capability to monitor, analyze online, and process various air and soil quality parameters associated with the condition of crops grown in greenhouses. The online monitoring, sample analysis and control techniques help to regulate various indices, balance nutrition level and maintain the crops' photosynthesis processes at its optimal level.

The designed system exhibits the following characteristics of IoT that allow it to meet the needs of precision farming.

- The system has incorporated cloud based web and mobile applications to enable online visualization of real-time streaming data. It is also equipped with data analytic functions for the examination of the data.
- The system does not get affected by occasional node failures caused by a variety of reasons, such as battery drainage, decay or hardware failure etc. of one of the intermediate nodes because the system has reliable transmission using acknowledgment of packets and retry counts.
- The developed system is scalable and easily deployable. It works as effectively for large deployments as it does for small ones, and can be expanded or reduced with minimal changes.
- The system is designed using power-efficient embedded devices and a communication module (i.e. ZigBee), which is the defacto standard for WSNs.
- The computational complexity of the sensor nodes during sensor data acquisition and routing is kept low.
- The system is easy to develop and debug as its design is based on an open source software platform.
- The system is designed to minimize the number of transmissions during routing by using data aggregation/fusion that is performed in the wireless sensor nodes.

1.5 Contributions

We developed a low-cost IoT based smart greenhouse monitoring system, which takes information about air and soil quality parameters inside the greenhouse and upload it to the web server, where it is accessible anytime and anywhere through internet. The system is smart because our various design approaches enable this system to create a self-forming, self-protecting routes within WSN, and self-monitoring environment with the help of internet based web server, which is aware of current status of the greenhouse and the possibility of raising an automatic alarm if the some unexpected event is generated.

The main research contributions of this thesis are summarized as follows:

- Our server, based on MEAN stack, accepts sensor data transported via the gateway node and keeps the same in the database, and enables faster access of data to analyze it from the descriptive data analytic services. Our data analysis techniques using past data are more detailed and useful to the users, as compared to other existing IoT based farming monitoring system. We deployed the system in the greenhouse and can monitor it remotely via both web server and mobile application.

- We also developed a multi-hop WSN in routing. The sensor node can form routes automatically to transport data to the gateway node. The sensor nodes are also capable of re-routing to an alternate address if any of the intermediate nodes fails. Our WSN system performs better in terms of reliability with many other traditional WSN routing methods. We validated the implementation after thorough testing, and did performance evaluation in terms of network setup time, latency, battery consumption etc.

- We also introduced an efficient cost metric for making routing decisions within WSN using hops count, and simple bi-directional link quality estimator using PRR and current battery voltage of neighbor nodes.

- We also implemented the methods which allow a new node to be integrated easily into the existing network, and a failed node be removed from the routing table. We validated the results successfully.

- We also implemented a new threshold-based event-triggering method of transmitting sensor data collected from nodes to reduce the number of transmissions within WSN to reduce energy consumption.

1.6 Organization of Thesis

The remainder of this thesis consists of nine chapters, including conclusions and future research. A brief description of each chapter is presented below:

Chapter 2 provides a brief description to the Internet of Things (IoT) and its overall functional architecture. It provides high-level design on how the IoT based functional architecture can be mapped on the design architecture of precision farming. It also provides a brief description of WSN and wireless communication standards.

Chapter 3 describes the hardware platform. It provides detailed hardware description of the sensors, the micro-controller unit, radio, mini-computer and other accessories, and how they are integrated together.

Chapter 4 describes the software platform. It provides detailed description of the open source platform, development environment, operating system, various tools, programming language required for the implementation of our system.

Chapter 5 describes the ZigBee protocol stack, its architecture and the serial protocols. It also reviews various routing protocols in WSN.

Chapter 6 surveys the previous research in IoT based smart monitoring projects and various experimental setups. It also reviews routing protocols in WSN.

Chapter 7 provides detailed description of the design choices available and made for the developed IoT based greenhouse monitoring system. It also presents our implementation of new routing algorithm and detailed programming methodologies, along with the flowcharts and the selected approaches.

Chapter 8 provides the results achieved during testing and performance evaluation of WSN routing. It also provides the web server data visualization and data analytic results.

Chapter 9 concludes this thesis and discusses future scope of research.

Chapter 2

IoT and WSN

2.1 IoT

The Internet of Things (IoT) is comprised by the integration of various technologies associated with the emerging wave of the internet-connected things which are part of the physical environment [5]. It is a system of objects, related digital and mechanical devices, actuators, various computing machines, sensors with unique identifiers and their ability to transport data over a distributed network without any interaction of man-to-man or man-to-computer. The IoT is an expansion, to the existing Internet.

With the rising wave of IoT technologies, the Internet will no longer be accessible only to humans, content, and mass communication but will also cover all real-world objects as smart devices sharing information, assisting business processes, communicating with people, and generating useful information. Some applications using IoT technologies have already being deployed. We can identify many examples of emerging application areas that are driven by different advantages, trends, and interests. Figure 2.1 shows how IoT technologies have been used in various applications, such as automotive transport, utility, smart city, farming, safety, health-care, by connecting people, vehicles, buildings, devices, industries, resources, and spaces together through Internet. IoT technologies are also used in remote monitoring and control applications. They are also used in compact and closed environments, such as factories and warehouses. When considering IoT technologies in closed environments, IoT can be termed as "Intranet of Things",

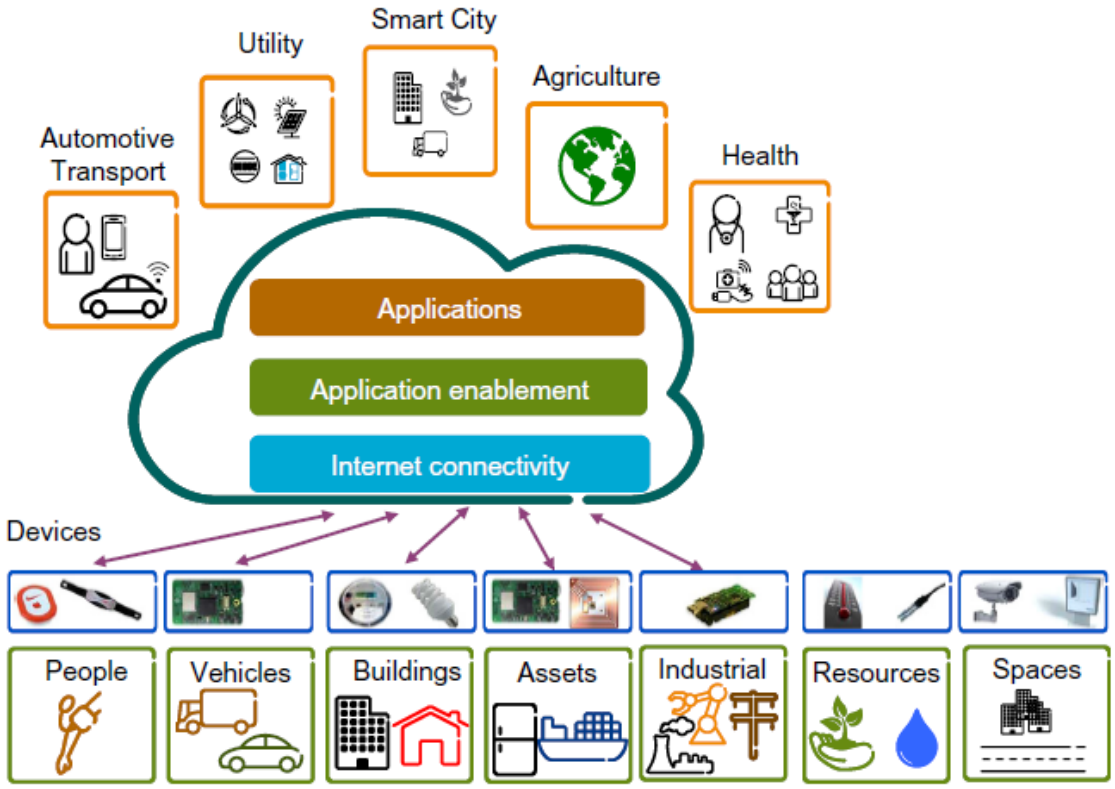

FIGURE 2.1: High-level IoT architecture and applications [5]

an optional solution of the IoT cloud [5]. IoT can be the enabler for having the novel promises made by various technologies.

As can be seen in Figure 2.1, the applications are very diverse, but we will focus on agriculture systems, and how IoT technologies are used in agricultural areas. More than 45 % of the world's population lives in cities and urban areas as of today, and already started looking for new IoT technologies that will yield more food production in their farming system, such as greenhouses. In fact, there is more inclination towards urban agriculture, such as greenhouse farming, compared to conventional cultivation. Urban agriculture, such as greenhouse, can be highly optimized using IoT technologies.

2.2 IoT Reference Architecture

Figure 2.2 shows the IoT reference architecture. The IoT reference architecture includes the important parts of IoT high-level architecture, such as designs fundamentals, capabilities, and protocol layers. This IoT architecture can be used as the foundation of building new IoT based applications, and can be further expanded and mapped into real

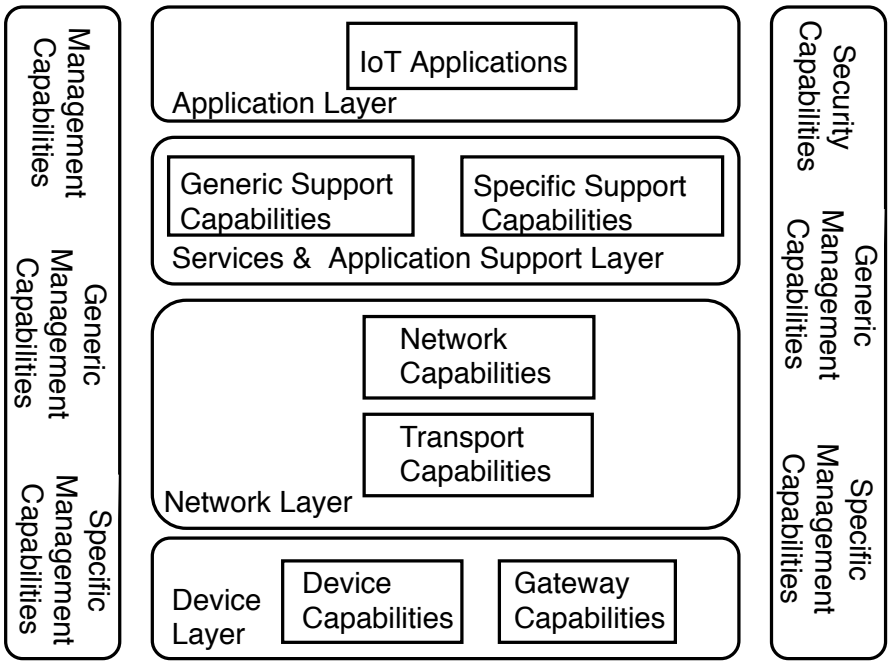

FIGURE 2.2: The reference architecture of IoT [5]

world entities by planning, implementing, and doing a thorough testing of the various modules of the real system.

The device layer includes the capabilities of the device and the gateway. The device capabilities include the interaction of devices directly within the network, as well as low-power operation functions; i.e. sleep and wakeup. The capabilities of gateway device include support for different protocols and protocol adaptation in order to make the pathways for the network layer capabilities and the device communication capabilities. The network layer provides networking capabilities, such as procedures for mobility management, and transport capabilities such as connectivity for the IoT data service. The service and application support layer consists of capabilities used by all IoT applications, such as data processing and data storage, and specific service capabilities fit to specific application domains, such as smart transportation or agriculture. The application layer provides the specific IoT applications. There is a broad array of different IoT applications, and typical examples include smart metering for the Smart Grid, vehicle tracking, environmental monitoring, factory automation etc.

2.3 Mapping of IoT Functional Model to Agriculture System

The functional model is defined as the combination of various functional groups (FGs) in a system. According to [5], the IoT functional model (FG) is the combination of various functional groups (FGs), such as device, networking and communication, system management, security management, IoT service functional, IoT service resolution, and IoT service organization. Each functional group associated with this model can be mapped into the design of agriculture system.

The devices functional group includes all the capabilities of the sensors, RFID tags, actuators, data storage etc. This includes sensing, processing, actuation, storage etc. While designing an IoT based farming system, we need to integrate different sensing devices together with micro-controller units and the communication modules, collectively known as a sensor node. Sensors are selected based on several air and soil quality parameters that impact plants' productivity in farming. The detailed description of the sensor devices that are used to build this system and their capabilities is covered in Chapter 3.

The role of the communication layer is to provide pathways for connectivity between the devices on one end and the different devices or the computing frameworks of the system on the other end. It also provides different communication standards to transport data over a distributed network. While designing an IoT based farming system, we need to carefully decide which communication standards we are going to use based on data rate, power-consumption and the communication range.

The IoT gateway protocol provides an end-to-end communication service connecting the gateway node to the IoT services using different protocols, such HTTP [6], CoAP [7], MQTT [8] etc. The choice of the gateway protocol is done based on reliability, security and simplicity of operation. As can be seen from Table 2.2, different gateway protocols use different transports, security protocols, QoS, addressing schemes, packet delivery assurance characteristics. In our greenhouse monitoring system, we plan to use MQTT to connect our gateway node to the server as it is more reliable than other protocols.

The security layer provides the functions that are required to maintain the information and management of the system secured and trusted. This can be accomplished by

Parameter	MQTT	CoAP	HTTP
Transport	TCP/IP	UDP/IP	TCP/IP
Paradigm	Publish/Subscribe	Request/Response	Request/Response
QoS	High	Moderate	No
Packet Delivery Assurance	High	Moderate	Moderate
Security	SSL/TLS	DTLS	SSL/TLS
RESTful	No	Yes	Yes
Addressing	Topic Based	URL	URL
Communication	Many-to-many	One-to-one	One-to-one

TABLE 2.1: A comparison of various protocols in IoT [6][7][8]

authenticating a user, providing authorization to access certain services, identification management system, exchanging keys between the client and the server, and using secure communication techniques to ensure the confidentiality of data that was exchanged. This also includes various encryption methods, and the needs of anonymity of the services.

We are using ZigBee for the communication within of the sensor nodes. ZigBee provides AES Encryption [9]. We need to set a 32 hexadecimal character string. When we enable AES encryption on one device, the data we send is encrypted. Similarly, the data we receive must first be decrypted in order to be in readable format. The encryption/decryption process can result in a slight increase in packet size and consumes more processing and memory of the sensor nodes. We are targeting a design for greenhouse for short reachability of the sensor nodes, and energy-efficiency of the sensor nodes, we do not include this feature.

The communication between the gateway node and the server can be secured using security protocols, such as transport layer security (TLS), datagram transport layer security (DTLS) [10] etc. TLS provides a transparent connection-oriented channel. However, it must run over a reliable transport channel, such as TCP because it is sensitive to loss of packets. The aim of DTLS is to introduce the some changes to TLS, and address this problem. The gateway protocol and web server based database are also secured with user id and password for authentication.

The IoT service functional group contains two functional components: IoT service and IoT service resolution. In IoT services, resources expose their features as services using

standard and open interfaces e.g. RESTful web services [11]. We need to have a agricultural database system which stores sensory data collected from various sensors. We can use either relational or non-relational databases. The interoperability between database service and the backend server can achieved RESTful web services. Resources can be hosted on the device memory itself or in the network as cloud database. If the resource is hosted on the sensor device itself, it can have the capability to answer the queries about the sensor readings; for example, it can respond to query to retrieve the temperature and humidity of an indoor or outdoor location. IoT service can fetch the queried information from the resources, accept the valid sensor data sent to the resources and store them. It also provides the subscription feature to retrieve data in an asynchronous fashion. IoT service resolution has the features required to contact IoT service. It can perform service-descriptors based discovery functions to make queries, enable look-up functions to users to access a service using service identifiers. The IoT service organization functional group acts as hub to connect many other functional groups. There are three components at this functional group; service orchestration, service composition, and service choreography. Service orchestration fulfills service requests that come from the users. The role of service composition is to fulfill the service requests coming from appropriate IoT service as well as other services. The service choreography acts as the broker to handle subscribe requests and publish messages. We have done via choosing IoT gateway protocols, such as MQTT, a publish and subscribe based protocol.

The system management layer contains the features associated with administrative rights, domain, configuration, setup related rules, state, membership, performance evaluation of system etc. The functions related to configuration related system enable to scale up the system to meet an increase the demands even during peak period.

2.4 Wireless Sensor Network Platform

A wireless sensor network is a distributed network comprising of many sensors and wireless modules to sense, observe and check the physical or environmental conditions, and transport their data through the network. These networks are bidirectional allowing not only the transportation of sensing data from source to the sink, but also transport control messages that have to take actions dictated by the processed information. A WSN might consist of a few to several hundreds or even thousands of sensor nodes with

each node connected to one or many sensors. Each sensing node has sensor device, processor, wireless communication module and power.

2.5 Wireless Communication Standards

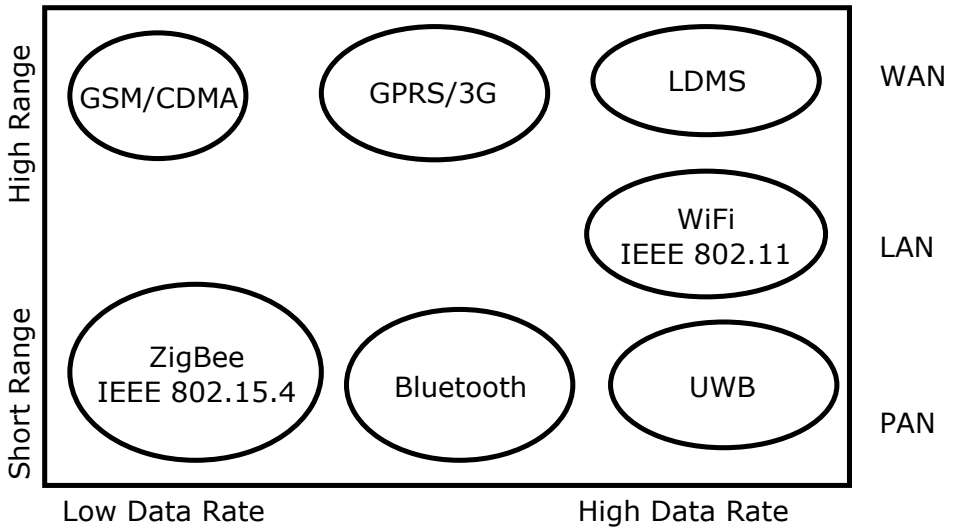

FIGURE 2.3: Wireless communication standards [12]

There are various wireless communication standards, such as IEEE 802.15.1, IEEE 802.11, IEEE 802.21, IEEE 802.15.4. IEEE 802.11, IEEE 802.15.1 families are prominent examples of WLAN and WPAN technologies. They have different communication range and data rate. IEEE 802.15.4 is the standard for different protocol stacks that targets IoT applications. ZigBee devices are based on standardized protocol architecture, which adopts IEEE 802.15.4 for its physical and link protocols. There is also a new protocol added to Bluetooth Low Energy (BLE 4.4) that aims to support IoT applications.

Chapter 3

Hardware Platform

The following hardware components are used as the design fundamentals of this system. They are Arduino UNO R3 [13], Raspberry Pi 3 [14], and various sensors, such as DHT11 [15], SHT10 [16], moisture sensor [17], carbon-monoxide sensor (MQ-2) [18], methane sensor (MQ-7) [19], light intensity sensor [20], XBee Pro S2 [22] etc.

In our case, a wireless sensor network is composed of several XBee devices as end nodes, several router nodes and a coordinator. The Raspberry Pi 3 is used as the gateway node and is not connected with the sensors. It is powered by an AC-to-DC adapter. It contains a micro-SD card with pre-installed NOOBS, USB port, XBee radio, and the CPU. NOOBS [23] is an installer to install Raspbian into Raspberry Pi.

3.1 Sensor Node

The sensor node has ATMega328P based Arduino UNO R3, the XBee-PRO S2 radio module, and various sensors, such as temperature, humidity, moisture, carbon-dioxide, carbon-monoxide, methane, light intensity, connected to it. The end nodes can be powered by 7 V - 12 V batteries or a USB power, and can be placed at any field location as long as they have XBee radios within the communication range. A photograph of the sensor node is shown in Figure 3.1.

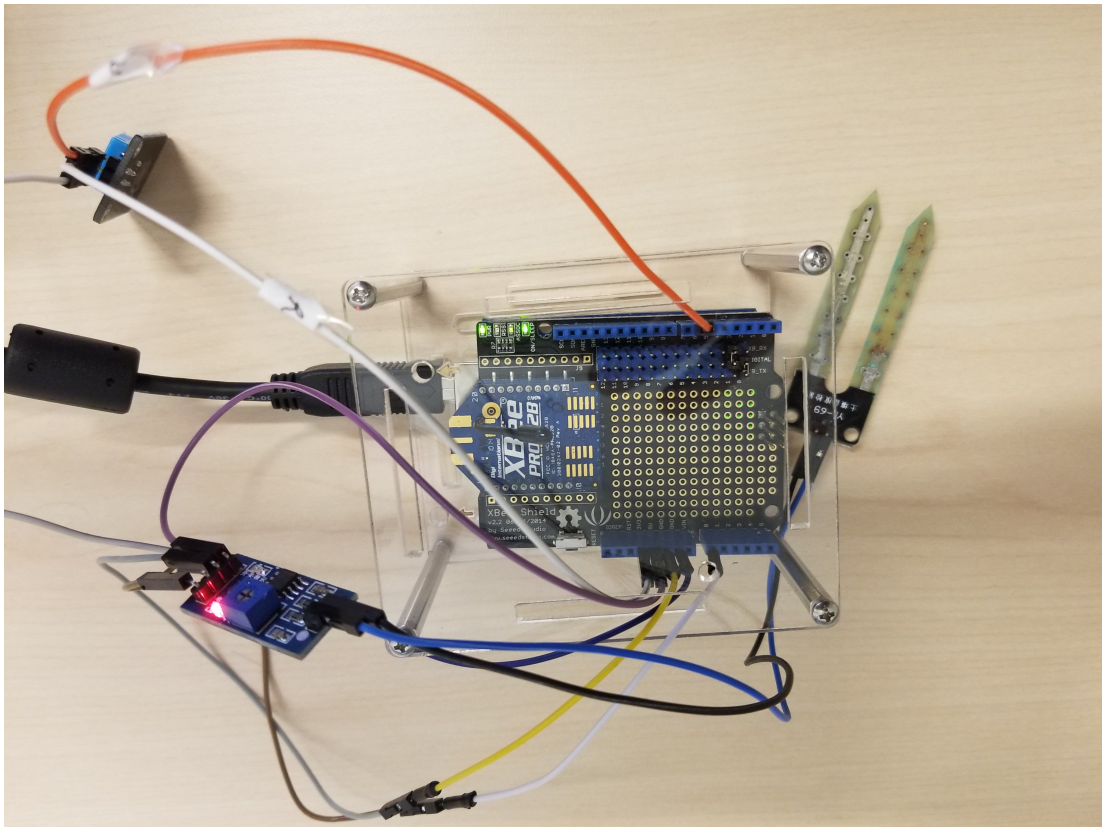

FIGURE 3.1: A sensor node

3.1.1 Arduino UNO R3

Arduino UNO R3 [13] has 8-bit ATMEL's ATMega328P micro-controller. It's operating voltage is 5 V, and runs at 16 MHz clock frequency. It can run up to 20 MHz at 5 V. It can be operated with 7 V - 12 V battery or DC power supply, but can tolerate input voltage up to 20 V because it has an internal voltage regulator. It has an ICSP header, a reset button, 14 digital I/O pins of which 5 pins provide pulse-width modulation (PWM) output, 6 analog pins (A0-A5) and can be used as an input or an output. Each digital I/O pin can draw 20 mA when the device is fully powered on. Arduino can generate 3.3 V and 5 V output voltage by the on-board regulator. The output voltage pin can draw maximum current up to 50 mA. The specification of Arduino UNO R3 is provided in the following Table 3.1.

Arduino UNO R3 is used as the development board for the sensor node because it is easily programmable using USB serial port instead of FTDI USB to serial driver chip.

FIGURE 3.2: Arduino UNO R3 [13]

MCU	ATMega328P
Operating Voltage	5 V
Clock Frequency	16 MHZ
Recommended Input Voltage	7-12 V
Input/output Pins (Digital)	14
Input/output Pins (Digital) PWM	6
Input/output Pins (Analog)	6
DC Current (per I/O Pin)	20 mA
DC Current (3.3V Pin)	50 mA
SRAM/EEPROM	2 KB/1 KB

TABLE 3.1: Arduino UNO R3 Specification [13]

3.1.2 DHT11

DHT11 sensor [15] is used to measure temperature and humidity. DHT11 gives both, analog and digital outputs calibrated with high precision values. It has 3 pins i.e. VCC, DATA and GND, as shown in Figure 3.3. A 10 kΩ pull-up resistor is connected between VCC and DATA pins. It provides a single serial interface of output data line (DATA) which can be easily read as a digital or an analog value by the micro-controller unit. It is small, responsive and consumes very low power.

3.1.3 SHT10

SHT10 [16] is used to measure temperature and relative humidity of soil. SHT10 is a product of Sensiron's family, and provides fully digital output. Band gap sensor is used for measuring temperature while relative humidity is measured by capacitive element embedded in it. It has 4-pins i.e. VCC, GND, DATA and SCK, as shown in Figure 3.4.

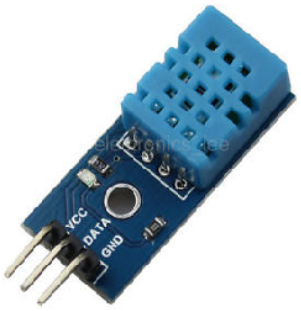

FIGURE 3.3: DHT11 [15]

FIGURE 3.4: SHT10 [16]

3.1.4 Soil Moisture Sensor

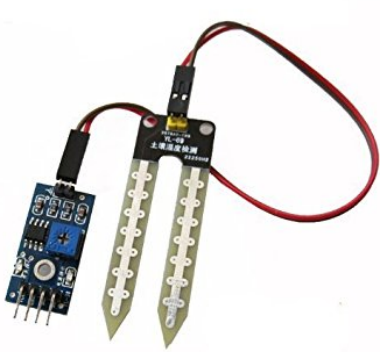

FIGURE 3.5: Soil moisture sensor [17]

Soil moisture sensor [17] is used to detect the level of moisture content in soil. As shown in Figure 3.5, it has 4-pins i.e. A0, D0, VCC and GND. They can be connected to Arduino via jumper wires. It has both analog and digital data outputs. It has the following components.

- Sensor blade for moisture level detection.
- Sensor comparator board based on LM393.
- High/Low Digital Output.

- Analog Output (0-1023).
- Mounting holes.

3.1.5 Gas Sensors

MQ-2 sensor [18] is used to detect methane (CH4) gas. It has 3-pins i.e. VCC, SIG and GND, as shown in Figure 3.6. It gives an analog output.

FIGURE 3.6: MQ-2 sensor [18]

MQ-7 [19] is used to detect carbon-monoxide gas from within the range 20 - 2000 ppm. It gives an analog output, as shown in Figure 3.7.

FIGURE 3.7: MQ-7 sensor [19]

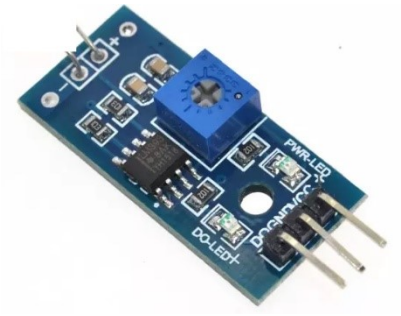

FIGURE 3.8: Light Intensity Sensor [20]

3.1.6 Light Intensity Sensor

Light intensity sensor [20] is to detect the level of light intensity. It consists of 3-pins; VCC, GND and SIG, as shown in Figure 3.8.

3.2 Gateway Node

A photograph of the gateway node is shown in Figure 3.9. It has Raspberry Pi 3, and the XBee Pro Series 2.

FIGURE 3.9: Gateway node

3.2.1 Raspberry Pi 3 Model B

The Raspberry Pi 3 [21] is a BCM2837 64-bit quad core ARM Cortex-A53 processor mini-computer operating at 1.2 GHz. It has 1 GB RAM, on-board Ethernet, Wi-Fi and Bluetooth 4.0. It has on-board 40-pin extendable GPIOs, 4 USB 2.0 ports, 4 pole stereo output and composite video port, as shown in Figure 3.10. It also has a DSI display port

for connecting the Raspberry Pi touch screen display, and a Micro SD port for loading Debian operating system and storing data. The specification of Raspberry Pi is shown in Table 3.2.

The Raspberry Pi 3 [21] is used as the gateway node. It is connected to the XBee-PRO S2 radio via a USB port. The gateway node is always powered on, and placed to a location that allow easy connectivity to internet. We have connected the Raspberry pi 3 to a WiFi router.

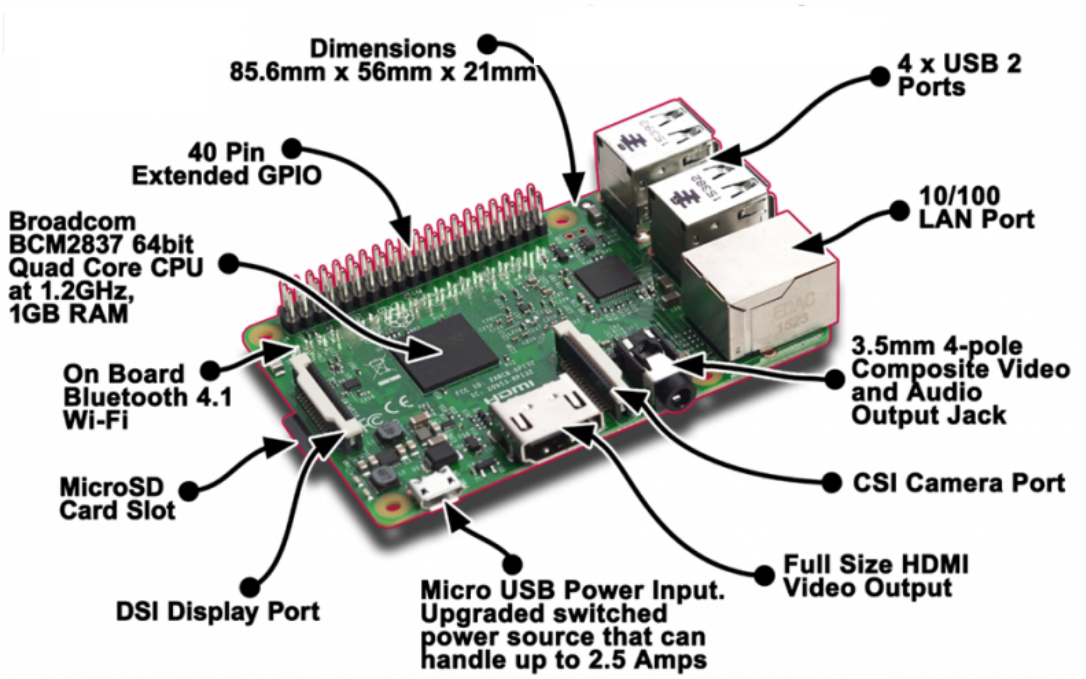

FIGURE 3.10: Raspberry Pi 3 Model B [21]

3.3 Radio Hardware

3.3.1 XBee-PRO S2

XBee-PRO S2 [22] follows the standard XBee protocol stack. They provide many features like, high-performance, low-power consumption, and the low-cost. Looking at these features, it is an excellent choice to build the sensor network. We used XBee-Pro S2 RF modules to design the wireless sensor network. They support point-to-point, point-to-multipoint, peer-to-peer topology. They can also establish mesh-networking. They

SoC	BCM2837
CPU	1.2 GHz 64-bit Quad Core ARM Cortex-A53 processor
RAM	1 GB
Operating System	Debian
Operating Voltage	5 V via micro USB charger
USB 2.0 ports	4
GPU	BCM2837 Video Core IV - 250 MHz
Video Input	CSI camera interface
Video Output	HDMI
Power Rating	1.5 W (300 mA)
On-board Network	Ethernet (10/100 Mbps), Wireless (802.11n), Bluetooth
On-board Storage	SD card slot

TABLE 3.2: Raspberry Pi 3 Model B Specification [21]

can be configured either in AT or API command modes using free XCTU software on a Windows machine. The specification of XBee Pro S2 is shown in Table 3.3.

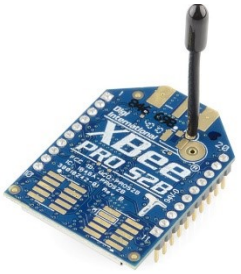

FIGURE 3.11: XBee PRO S2 [22]

3.3.2 XBee-PRO S2 Arduino Shield

XBee can be mounted on the XBee Arduino Shield. The connection settings need to be according to Table 3.4 in order to establish TX-RX communication between Arduino, and the XBee radio module. It connects the TX pin of Arduino to the RX or DIN pin of XBee, and the RX pin of Arduino to DOUT or TX pin of XBee, as shown in Table 3.4. This can be directly connected using jumper wires between Arduino and the XBee, according to Table 3.4. Fortunately, there are different XBee shields available in the market that make this communication convenient. We used XBee shield for Arduino [24]

Indoor Range	90 m
Outdoor Range (Line of Sight)	Up to 1 mile
Power (Transmit)	63 mW
Receiver Sensitivity	-100 dBm
RF Data Rate	250, 000 bps
TX Peak Current	250 mA
RX Current	40 mA
Power Down Current	Less than 10 μA
Communication	Uni-cast/Broadcast
Interference Immunity	Direct Sequence Spread Spectrum
Channel Access	CSMA - CS
Operating Voltage	2.8-3.4 V
Operating Frequency	ISM 2.4 GHz
Encryption	128-bit AES encryption

TABLE 3.3: XBee-PRO S2 Specification [22]

shown in Figure 3.12, provided by Seed Studio to mount the XBee, because it extends all the pins of Arduino. With XBee mounted on this shield, TX-RX communication between Arduino and XBee can be done easily with simple jumper settings. The jumper settings can be set either in USB or XBee mode. After required settings are configured via the XCTU software, the jumper setting should be in XBee mode so that it starts listening or sending packets.

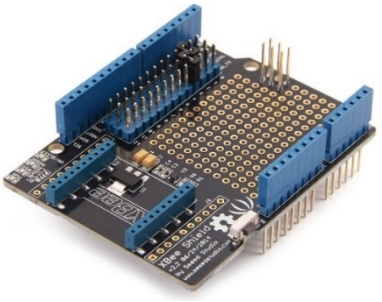

FIGURE 3.12: XBee-PRO S2 Arduino Shield [24]

3.3.3 XBee-PRO Explorer Dongle

XBee can be put in programmable mode directly using Arduino in bootloader mode with the settings described in Table 3.5. Arduino can enter into bootloader mode by

Arduino	XBee
TX or 1	DIN or RX
RX or 0	DOUT or TX
VCC or 3.3 V	3.3 V
GND	GND

TABLE 3.4: Connection Settings between Arduino and XBee

connecting RESET pin of Arduino to GND bypassing the ATMega328P unit in it. This setting is inconvenient whenever we need to load the new firmware frequently during the development phase. Moreover, recent version of XCTU software needs the XBee module to reset during flashing. Fortunately, there are XBee Explorer Dongles available in the market with RESET button. A RESET button can be seen in Figure 3.13.

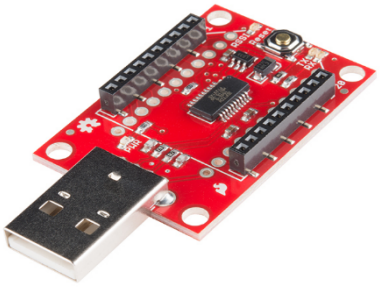

FIGURE 3.13: XBee-PRO Explorer USB Dongle [25]

An XBee Explorer Dongle is used for many purposes. It has a FT231X USB-to-Serial converter, which translates data between the XBee and the computer. There are 4 LEDs that will indicate different operations carried by the XBee i.e. TX, RX, RSSI (received signal strength indicator), and a power indicator, which helps in debugging. With this Dongle, we can directly mount the XBee on it, plug the unit to our computer's USB port, and load the firmware using XCTU software. No connecting wires are needed. It also has an on-board voltage regulator, which tolerates up to 500 mA current.

Arduino	XBee
TX or 1	DOUT or TX
RX or 0	DIN or RX
VCC or 3.3 V	3.3 V
GND	GND

TABLE 3.5: Setting to put XBee in programmer mode via Arduino

The XBee dongle is also connected to the Raspberry Pi via a serial port. We need to specify the correct serial port while writing the code to read serial packet in python.

3.4 Power Supply and Other Components

- Batteries, AC-DC adapter or USB cables, LED and connecting wires, 1 KΩ, 100 Ω, 10 KΩ resistors

Chapter 4

Software Platform

Every sensor node requires a development environment for programming. The gateway node requires a software, and an operating system to run. The web server and mobile applications require communication between the gateway node, and a web server, or a cloud server via the communication protocol. More about IDE, software and OS have been discussed in the following sections.

4.1 Arduino IDE

A micro-controller unit requires a development environment for programming before it can be ready for use. We need to write, build and load the source code using a development environment into micro-controller's flash memory. We used the latest version of open source cross-platform supported; Arduino IDE version 1.8.1 [26], provided by Arduino, for our development work. It can be downloaded for free. Arduino IDE [26] is supported across many platforms e.g. Windows, Linux, MAC etc. It provides programming support of C and C++. Sensor node is programmed in C. We need to include C++ libraries required by source code. We used open source libraries for various sensors, such as DHT11 [27], SHT10 [28]. Arduino IDE also provides a serial debugging monitor which is useful for debugging.

4.2 Raspberry Pi 3 Platform

The gateway node should have robust hardware and faster processing processor than a sensor node because it is involved in more processing than other sensor nodes. The gateway node is always powered on and located near to place where access to the internet is possible. Nowadays, even small size gadgets (e.g. Android based smart phones) run Linux based operating system. Linux is highly customized and free open source operating system. There are different distributions of Linux e.g. Red Hat, Fedora etc. The Raspberry Pi 3 as the gateway node is not suitable for all of them, as not all of them are light-weight. Fortunately, there is a customized Linux to make it light-weight, and an easier installation process.

Before the development of Raspberry Pi 3 version, NOOBS [29] required to be installed manually by downloading it for free from the Raspberry Pi website, and copying into SD card. NOOBS (New Out of Box Software) [29] is an easy operating system (OS) installation for Pi 3 which allows to install different operating systems on it. We used Raspberry Pi 3 (Model B) [30] as it comes with pre-installed NOOBS on SD card, which installs Raspbian in it. Raspbian is a Debian based operating system. It is is free and optimized for the Raspberry Pi hardware. The Raspberry Pi 3 supports HDMI input which can be connected to any HDMI output supported monitor, and can be used as a desktop computer. The Raspberry Pi 3 comes with a default configuration of SSH, as "disabled" because of security issues. So, we require to enable SSH from the configuration menu of Raspbian. Once it is enabled, we can access it remotely by simply connecting from putty terminal as shown in Figure 4.1, and do not need to be dependant on HDMI monitor. Alternatively, we can access the Pi 3 computer via Virtual Network Computing (VNC) desktop server. Once Pi 3 is ready for outside field use, we can easily disable SSH again to avoid security issues.

We implemented the gateway side code using python script. Raspbian comes with built-in python. We do not require to install python again, but require installing pip to install packages. We used the latest version (9.0.1) in order to install python packages.

Following python packages are installed.

- paho mqtt client in oder to install MQTT client to publish MQTT topic and payload to the server.

- Http package.
- XBee 2.3.1: XBee API messages can be accessed using XBee 2.3.1 package only.

4.3 XCTU

We need to set the correct function sets, and the configuration parameters as described below for the XBee before it is ready to use. XCTU [31] is an open source software provided by Digi International to upload the new firmware of the XBee, and set such parameters. According to the Digi International website [31], it is supported only on Windows PC so far. We can set various parameters via XCTU after mounting the XBee on the XBee adapter dongle, and connect to USB port of any computer.

- Function Set: It facilitates whether to use the XBee as a coordinator, router or an end device. It gives an option to choose AT, API or I/O mode.
- Networking:
  ID PAN ID: It allows to set the unique PAN ID. The same ID should be used by all devices in the network.

  SC Scan Channels: The support of number of channels depends on the types of XBee radio (i.e. Series1 or Series2). The default SC Scan Channel is 0xFFFF for Series 2.

  NJ Node Join Time: It is defined as the time required to join the node within a network.

  NW Network Watchdog Timeout: If this value (NW) is set greater than 0, communication from the coordinator is continuously monitored by the router node, and leaves the network if it can't communicate with the coordinator for a certain period. By default, this value is 0.

  JV Channel Verification: The power-on join verification check is enabled with join verification (JV) value. With JV set to 0, the router device continues operating on its current channel number even if a coordinator device is not detected. With JV set to 1, a node will first try to discover the 64-bit address of the coordinator when it tries to join the network for the first time.

JN Join Notification: If JN is enabled, indicates that a device has joined the network.

- Addressing:

  Serial Number High and Serial Number Low (SH High/SH Low): Every XBee device has a unique 64-bit address. The sender can set SH high and SH low (i.e. higher and lower bytes address) to specify its own source address.

  Destination Address High (DH) and Destination Address Low (DL): The sender can specify the 64-bit address (i.e. higher and lower bytes destination address) of the receiver when it wants to send data to another XBee. The default value is 0/0.

- Baud Rate: The baud rate is the speed at while serial data transmits. The XBee is configured to set the baud rate of 9600 bps.

4.4 MEAN Stack Based Web Server

The MEAN stack based web server application is developed using Node.js [32], MongoDB [33], Express.js [34] as back-end system. HTML5 [35], jQuery [36], AngularJS [37], Bootstrap and CSS are used as front-end client UI.

The brief description of each framework is given below.

4.4.1 Node.js

To develop web server applications, we need to have run-time environment. Node.js [32] provides a run-time environment for executing server-side scripts. It is an open source platform. It provides run-time environment that work across many platforms. It uses an event handling, non-blocking I/O model, that makes it efficient and light-weight. It provides the biggest pool of open source packages known as 'npm'. 'npm' includes many packages, such as express, http, socket.io, mongoose etc.

4.4.2 MongoDB

To store IoT sensor data, we need to have database service running which interacts with the back-end system. There are different kinds of database; i.e. relational or document

oriented database. We chose MongoDB over relational database. MongoDB [33] is a free, and an open-source cross-platform database. It supports JSON based query language with schema. It is classified as NoSQL database, and is comparatively faster and scalable. It is faster because it does not need to maintain the relation between the tables.

MongoDB [33] offers data aggregation model using the concepts of data processing pipeline called aggregation pipeline framework. The aggregation pipeline is very useful for the descriptive data analytic services to compute hourly, daily, weekly, monthly and yearly average, maximum, minimum values of data.

4.4.3 Express.js

Express.js [34] is a server-side framework based on Node.js. The framework allows to set up middle-ware to respond to HTTP requests. It defines a routing table, which is used to perform different actions based on HTTP.

4.4.4 HTML5

HTML [35] is used for developing web client UI. It provides several components such as tags and their attributes, which makes it distinguishable from the text.

4.4.5 jQuery

jQuery [36] is fast and small JavaScript based library which offers lots of useful features. Things like HTML document traversal, event loop handling, animation, and Ajax calls are much simpler and efficient using these features. It also offers an easy to use API that works efficiently across multiple browsers.

4.4.6 AngularJS

AngularJS [37] is a modern JavaScript based application framework. HTML attributes can be easily extendable with AngularJS which provides the directives and binds data to HTML.

4.4.7 Socket.io

Socket.io [38] is a JavaScript library mainly, applicable for real-time web applications. It enables real-time, as well as bi-directional communication between web clients such as browsers and the servers. It mainly consists of two parts; a browser running, and Node.js server running in the server-side. Socket.io is event-driven like Node.js. It has similar APIs like Node.js. Socket.io primarily can be used as simply a wrapper for web socket. It provides many more features like broadcasting to multiple sockets, storing data associated with each client, and asynchronous I/O.

4.5 Thingspeak Cloud Based Web Server

The advantage of using "Thingspeak" cloud based URL is: users do not have to be maintaining the database. Thingspeak [39] maintains SQL database in the cloud, and provides access to retrieve time series data using simple get JSON data interfaces. The time-series data can be easily accessed using Android framework for "Thingspeak".

We can post the data to the thingspeak cloud server "http:api.thingspeak.com:80" or publish to "mqtt.thingspeak.com". We can retrieve the data from thingspeak URL using "GET JSON" interface functions.

4.6 Android Studio IDE

Android is an open source Linux based operating system, and Android Studio is the official IDE to develop Android applications. The developed Android application is based on "thingspeak" [39] cloud. Thingspeak [39] is an open source platform to store IoT data in the cloud database. Sensor data can be reported to "thingspeak" cloud by posting HTTP requests or MQTT publish. It internally maintains the SQL database. We can retrieve a time series data from its URL through on a public channel, read/write API and different fields (e.g. Field 1: Temp; Field 2: Humidity). To build "thingspeak" cloud database, we need to create an account on "thingspeak.com" [39]. This account is free. It gives us different read and write API keys, and a public channel number. Then we need to create different fields. Fields are basically different data types; e.g. temperature,

humidity, water level etc. with the same names that are used while posting the data from Raspberry Pi. The application is configured in such a way that if we launch the application from a mobile and synchronize to the same channel number, API keys and the fields to retrieve data from the cloud, we can visualize data on real-time as widgets.

Android Studio 2.3.3 [40] has been used to develop a weather widget. The output is mentioned in Chapter 8.

Chapter 5

ZigBee Networking Principle and Serial Protocols

In this chapter, we will describe ZigBee networking, the ZigBee protocol stack, and related serial protocols. We will describe the frequency bands ZigBee operation in, channels, data rates, and supported modulation schemes. We will present ZigBee protocol stack, the role of each layer, IEEE 802.15.4 device types and roles, network topology.

5.1 The ZigBee Wireless Networking

The XBee module we used operates at the ISM band, located at 2.4 GHz frequency range. The modulation scheme is O-QPSK. The ZigBee standard has 16 channels (within the range of 2400 - 2483.5 MHz). ZigBee can operate in 868 MHz, 915 MHz and 2.4 GHz depending on its types (i.e. Series1 or Series2), and can provide maximum data rate of 250 kbps [41]. Different supported frequency bands, channel, modulation, data rate are mentioned in Table 5.1.

5.2 ZigBee Protocol Stack

The ZigBee protocol stack [42] consists of four basic layers; physical, medium-access control, network, and application layer as defined in OSI model. It is necessary to mention the IEEE 802.15.4 standard ZigBee used in and developed for PAN. The standard

Physical Layer	Frequency Band	Channel	Digital Modulation	Data Rate	Geographical Area
868-915 MHZ	868-870 MHZ	0	BPSK	20 kbps	Europe
868-915 MHZ	902-928 MHZ	1-10	BPSK	40 kbps	America
2.4 GHZ	2.4-2.4835 GHZ	11-26	O-QPSK	250 kbps	Worldwide

TABLE 5.1: Frequency bands, channel number, modulation, and date rate of ZigBee [41]

allows the user to define upper layers of the OSI model. The physical and medium-access layers are covered in the IEEE 802.15.4 WPAN standard, and the network and application layers are covered in the ZigBee alliance supported documentation.

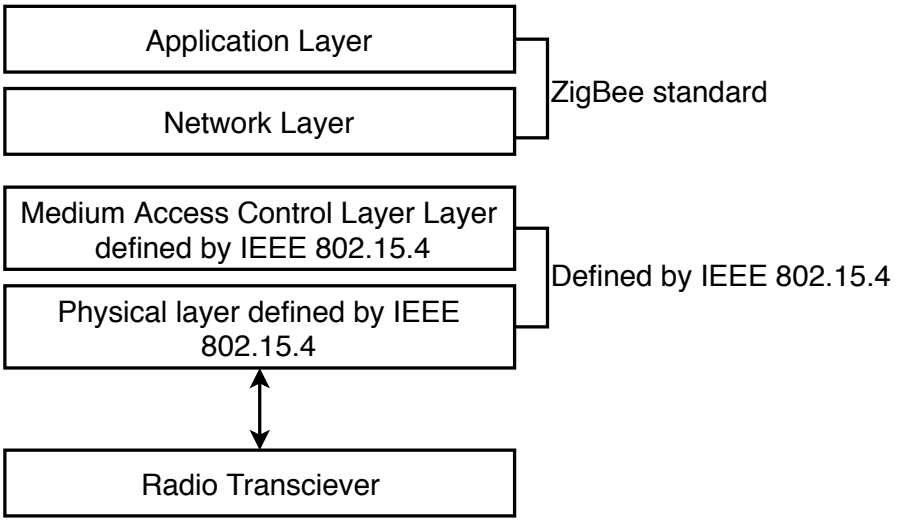

FIGURE 5.1: ZigBee protocol stack [42]

5.2.1 The Physical Layer

The role of physical layer includes the activation of the radio hardware that transmits, and receives the packets via a wireless channel. It also includes the selection of currently unused channel, and directly controls and communicates with the radio transceiver. It defines 3 different frequency bands and 27 channels of 5 MHz each.

The physical layer packet contains three parts i.e. synchronization header, physical Header, and the physical payload as shown in Figure 5.2. The synchronization header (SH) contains preamble of 4 bytes, and SFD of 1 byte. The physical header contains a frame length of 7 bits, which contains reserved 1-bit. The physical payload has variable

payload. The preamble has 4 bytes (32-bits) size, and is used for chip and symbol synchronization at the receiver. Start of frame data (SFD) is of 8-bit field, which is used to segregate between preamble, and actual data of the physical layer. The frame length is 7-bits long and defines the total number of octets contained in the payload field. PSDU is the actual physical data, and its size is variable.

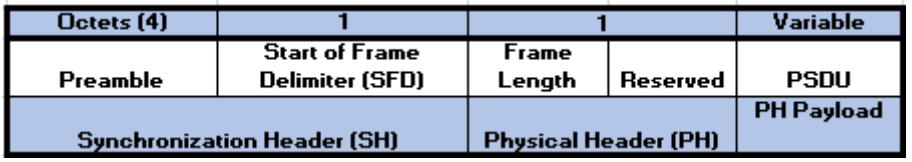

FIGURE 5.2: Physical layer header format [42]

5.2.2 The Medium-Access Layer

The MAC layer is located between the network and physical layer. MAC layer's responsibilities include the generation of beacon packets, synchronization of beacon packets, association, and the dissociation of packets.

The generic MAC frame consists of MAC header, MAC payload, and MAC footer, as shown in Figure 5.3. The MAC header contains information, such as frame control, sequence number, PAN ID, source and destination address. The MAC payload contains the frame payload, and the MAC footer consists of frame control sequence (FCS). It has frame control field of 2 octets, and consists of useful information such as frame type, security-enabled, ack-request, intra-PAN, source, and destination addressing mode. It defines 4 types of MAC frames; data frame, beacon frame, ACK, and MAC command frame, and the type is specified by 3-bit field (i.e Frame Type). The beacon frame consists of the frame control field, addressing fields, and sequence number. The beacon frame is broadcasted often by the coordinator device in order to obtain PAN ID of its nearby devices. It is also used to synchronize clocks within the network. The data frame is used for transmitting data. The acknowledgment frame is used to inform the successful reception of a transmitted frame. The MAC command frame is used to transmit MAC command frames.

Octets (2)	1	0/2	0 2 8	0/2	0 2 8	Var	2
Frame Control	Sequence Number	PAN ID Destination	Destination Address	PAN ID source	Source Address	Frame Payload	FCS
MAC Header						MAC Payload	MAC Footer

FIGURE 5.3: Generic MAC frame [42]

5.2.3 The Network Layer

The network layer is located between application layer and MAC layer. The network layer frame consists of network header and network payload. Its responsibility includes routing and formation of network. Routing involves the selection of route from source to destination. The packet is delivered via this route. In ZigBee, a network is composed of a coordinator, routers, and several end devices. The coordinator and router perform path-discovery and do the maintenance of paths in the network. The network layer's functions are; establishing a new network, selecting a topology and providing network addresses to the other devices within the network.

5.2.4 The Application Layer

The application layer is the top layer in the protocol stack of ZigBee. The application layer's responsibility includes holding application objects i.e. to control and manage protocol layer in a ZigBee device. Separate set of application profiles can be built on the application layer. Application profiles are built using application specific packet formats and should be compliant to ZigBee packet frame structure and standard. This compliance ensures interoperability when products from different vendors interact each other.

5.3 IEEE 802.15.4 Device Type and Role

There are two types of device as per IEEE 802.15.4; full function device and reduced function device. Full function devices have the capability of talking to all other devices within a network, whereas a reduced function device can talk only to full function devices. In ZigBee, there are three types of device; coordinator, router and end devices. In a ZigBee network, there is one coordinator, a few routers, and many end devices.

- Coordinator: Coordinator acts as full function device. It starts a network formation request and allows routers and end devices to join the network, transmit and receive data as well as route the data. There must be a coordinator in a network.
- Router: Router also acts as full function device, which has the capability to relay data and allows other devices to join the network.
- End device: End Devices are basically reduced function devices, having limited capabilities. They can not perform the routing of messages. Their role is to only transmit to, or receive message from other routers or the coordinator. These are typically sensor nodes which send or receive.

5.4 Network Topology

Network topology is managed and maintained by ZigBee's network layer. IEEE 802.15.4 gives star, tree, cluster tree, and mesh topology as shown in Figure 5.4 [42]. ZigBee supports only star, tree, and mesh topology.

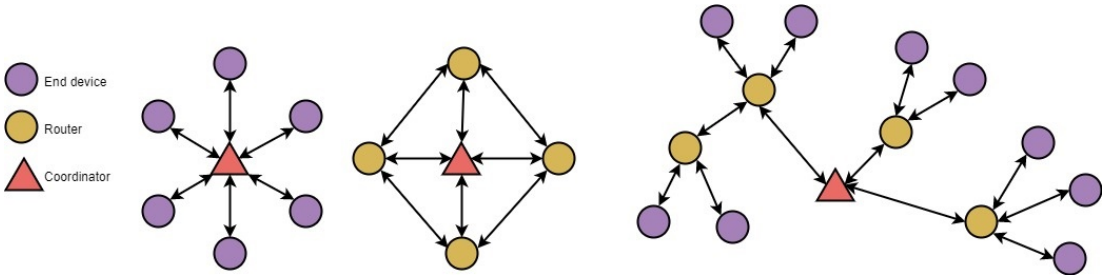

FIGURE 5.4: a. Star, b. Mesh c. Tree Topology [42]

5.4.1 Star Network

Within star networks, the coordinator is the central node, which handles the requests coming from various sensor nodes.

The characteristics of star networks are as follows:

- With star topology, end devices are in communication only with the central node i.e. coordinator.
- There is a coordinator involved in any packet exchange between end devices.

- Coordinator can get congested because all the end devices are reporting to a single coordinator.The entire operation of a network depends on the coordinator. This is disadvantage of star network topology.
- There is no alternative pathway from the source to the destination. The advantage of star networks lies in its simplicity of operation. Source packets go through two hops at most to reach its destination. It can be used in small networks.

5.4.2 Tree Network

In tree topology, the network consists of a central node (i.e. a root node), which is the coordinator, several routers, and the end devices. The function of router is to extend the network coverage. End nodes that are connected to a coordinator or routers are called children. Only routers and the coordinator can have children. Each end device is only able to communicate with its parent (i.e. router or coordinator). The coordinator and routers can have children and, therefore, are the only devices that can be the parents. The end device cannot have children and, therefore, may not be the parent. A special case of tree topology is called cluster tree topology. The disadvantages of tree topology are that if one of the parents become disabled, the children of the disabled parent cannot communicate with other devices in the network; even if two nodes are close to each other because they cannot communicate directly.

5.4.3 Mesh Network

The mesh topology, also referred to as a peer-to-peer network, consists of full function devices; one coordinator and several routers as shown in Figure 5.4 (b). A mesh topology is a multi-hop network i.e. a packet passes through multiple hops to reach its destination. The range of a network can be increased by adding more devices to the network. A mesh topology is self-healing, meaning during transmission, if a path fails, the node will find an alternate path to the destination. Devices can be close to each other so that they use less power. Adding or removing a device is easy. Any source device can communicate with any destination device in the network. Compared with star topology, the mesh topology generates larger overhead. Mesh routing uses a more complex routing protocol than a star topology.

Both tree and mesh topology allow the use of ZigBee routers to extend communication at the network level.

Following are the operations done by a coordinator irrespective its network topology:

- Allocating a unique 16-bit or 64-bit address to each device within the network.
- Initiation, routing and termination of the messages throughout the network.
- Selecting a unique PAN ID for the network.

5.5 XBee Serial Protocols

There are two modes supported for serial communication i.e. AT mode (transparent mode) or API mode [43].

5.5.1 AT Mode

AT mode is the simple version of serial communication. It is the replacement of serial line in which all UART data get queued up for radio transmission and finally sent out via the DOUT pin when it is ready. In AT mode, XBee can be configured to send data, only to a specific remote device or broadcast to all the devices (0xFFFF). The destination address should be specified during uploading the firmware. We can not change destination address from the host side.

5.5.2 API Mode

In API mode, all transmitting or receiving data is placed in frames, which define different commands or operations within the system. The features of API mode are as follows:

- It is possible to set module and routing at the host side from the application layer using API mode. In addition to this, we can configure to set a single or multiple destination addresses to route the data to a specific or remote device from application side itself.

- API mode is faster as compared to AT mode. In AT mode, the application enters AT command mode, change the address, exit the command mode and then send data.
- API mode transmission returns a status response after sending frame which indicates success or failure with reason.
- The broadcast as well as multicast transmission are supported.
- Integrity of data packets is maintained with the checksum value.
- With the remote AT feature, remote radio can be configured.
- Received packets can be identified by source address present in the frame.

Data is transmitted and received in a defined format or structure when it is communicated using API mode of communication. API mode data frame is implemented as UART data frame. Two types of API frame modes are defined; AP=1, AP=2.

API Frame Data: Different application specific frame structures have been defined by XBee. Every application defines its own API identifier.

ID	API Frame
0x08	Immediate AT CMD
0x09	Queued AT CMD
0x88	AT CMD RES
0x8A	Modem Status
0x10	ZigBee TX REQ
0x8B	ZigBee TX Status
0x90	ZigBee RX PKT
0x17	Remote CMD REQ
0x97	Remote CMD RES

TABLE 5.2: XBee API Frames [43]

5.6 XBee API Frames

We use the following set of API frames in our design; XBee Transmit Request, XBee Transmit Status, XBee Received Packet.

5.6.1 XBee Transmit Request

Frame Field	Byte Value/Offset	Description
Start Delimiter	0x7E (1)	Indicates the beginning of the data frame.
MSB	0x00 (2)	Most significant byte (MSB) of the length of the data payload.
LSB	0x16 (3)	LSB of the length of the data payload.
Frame Type	0x10 (4)	It determines which type of API frame (i.e. for transmit request, the value is 0x10) is used.
Frame ID	0x01 (5)	It determines API frame for the application to correlate with a subsequent ACK. If this value is set to 0, response is not received from the device.
64-bit Destination Address	0x00 (6) 0x13 (7) 0xA2 (8) 0x00 (9) 0x40 (10) 0x0A (11) 0x01 (12) 0x27 (13)	It sets the 64-bit address of the remote device.The order is: MSB first and LSB last. Below is 64-bit address reserved for coordinator and broadcast. 0x0000000000000000 (coordinator) 0x000000000000FFFF (broadcast)
16-bit Destination Address	0xFF (14) 0xFE (15)	It sets the 16-bit address of the remote device. The default value is 0xFFFE.
Broadcast Radius	0x00 (16)	It defines the number of hops count value. If this value is set to 0, maximum number of hops count value is considered.
Options	0x00 (17)	This is optional parameter and sets different properties within API frame. 0x01 = Disable retry. 0x20= Enable AES encryption field if EE=1. 0x40= Extended TX timeout value is enabled by XBee stack.
RF Data	0x54, 0x78, 0x44, 0x61, 0x74, 0x61, 0x30, 0x41, 0x13 (Bytes 18-to- n)	This field contains all the data sent from source to the remote device.
Checksum	0x13 (n+1)	Data integrity check is enabled with checksum value.

TABLE 5.3: Detailed description of XBee transmit request frame [43]

The data format of XBee transmit request frame [43] is described in Table 5.3. The byte offset at 1st, 2nd, 3rd and last resemble frame structure as same as generic API frame i.e. byte 1 is used for 'Start Delimiter', byte 2 is MSB of the length of the payload, byte 3 is LSB of the length of the payload and the last byte is for 'checksum'. In addition

to this, XBee Transmit request frame contains frame type as 4th byte. It is used to set which type of API Frame is used. The 5th byte contains the frame ID which helps the application to correlate frames between sender and its receiver.

It also has bytes 6-13 reserved for 64-bit remote device address i.e. bytes 6-9: MSB of the remote address and bytes 9-13: LSB of the remote address. If the remote address to be set is only Coordinator, bytes 6-13 can be set as 0x0000000000000000 and if it is to be broadcast, then it can be set as 0x000000000000FFFF. Bytes 14,15 can be set for placing the 16-bit destination address. The default value of this byte is 0xFFFE.

5.6.2 XBee Transmit Status

The data format of XBee transmit status frame [43] is described as per Table 5.4. The XBee transmit status frame gives the advantage of returning transmission status while sending some data. The length of this frame is 11 bytes. Byte 1 is 'Star Delimiter'. Bytes 2 contains the most significant bytes of the length of the payload. Byte 3 contains the least significant byte of the length of the payload. In addition to this, this API frame format contains 'Frame Type', the 4th byte, which is used to set which type of API Frame being used. The 5th Byte- 'Frame ID' helps the application to correlate frames between sender and its receiver. Bytes 6, 7 sets the 16-bit address. Byte 8 is to store transmission retry counts. Byte 9 is used to specify the status of the delivery. Byte 10 is used to collect the discovery status. The few response codes value are shown in Table 5.5.

5.6.3 XBee Received Frame

The XBee received frame contains received data starting from bytes 16 to n depending upon the size of the data payload. The "Options" field sets whether packet was acknowledged, broadcast, encrypted with AES and sent from end device with code 0x01, 0x02, 0x20, 0x40 respectively. 'Frame Type' used for this API frame is 0x90.

Frame Field	Byte Value/Offset	Description
Start Delimiter	0x7E (1)	The start of the frame.
MSB	0x00 (2)	Most significant byte (MSB) of the length of the payload.
LSB	0x16 (3)	Least significant byte (LSB) of the length of the payload.
Frame Type	0x8B (4)	Determines which type of API frame is used.
Frame ID	0x01 (5)	Determines API frame for the application to correlate with a subsequent ACK. If set to 0, the device does not send a response.
16-bit Destination Address	0xFF (6) 0xFE (7)	Sets the 16-bit address of the remote device. The default value is 0xFFFE.
Transmit Retry Count	(8)	Sets the number of retries in transmitting frame.
Delivery Status	0x00 (9)	Indicates the status of delivery
Discovery Status	0x01 (10)	Indicates the status of discovery

TABLE 5.4: Detail description of XBee transmit status frame [43]

Description	Code
Success	0x00
Acknowledgment Failure for MAC	0x01
Failures in Clear Channel Assessment (CCA)	0x02
Destination endpoint invalid	0x15
Acknowledgment Failure for Network	0x21
Not Joined to Network	0x22

TABLE 5.5: Response codes of XBee transmit status [43]

Chapter 6

Literature Review

6.1 Introduction

In this chapter, we will discuss the state-of-the-art of the technology areas, which are at the core of the system design we have performed. It has been organized in different sections. At first, we provide some of the previous work in smart agriculture systems. The objective here is to introduce some related research work, and describe the advantages and the drawbacks of these methods. We will then provide a summary of various experimental setups of WSN, and compare the test-bed setup of our design with the existing ones. We will then describe various methods of the data handling and streaming mode. Lastly, we will review traditional routing protocols and link quality estimators in WSN, and summarize their limitations.

6.2 Related Work in Smart Agriculture

The significance of experimental research in WSN has been acknowledged by various research communities, and used in many areas including the support and service of devices embedded system.

A previous attempt to design a WSN based smart agriculture system was made by Zhou, et al. [46]. It comprises of a wireless sensor node, an environmental web server station, many wireless actuators, and a controller, as shown in Figure 6.1. The sensor node senses and collects data within its coverage area. A nearby web server station continuously

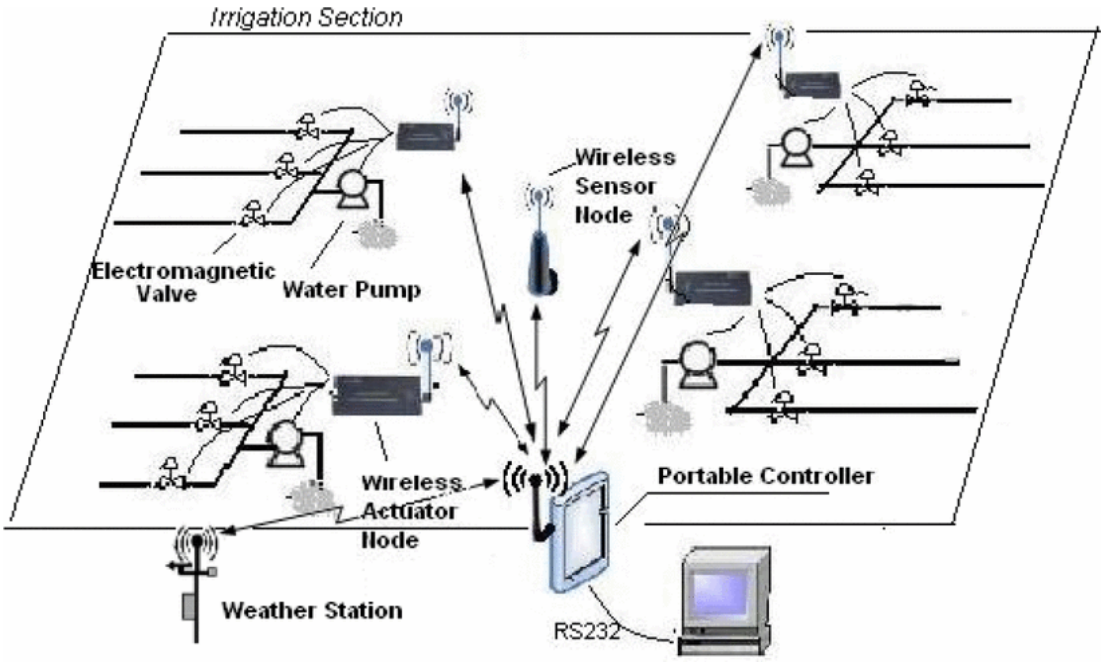

FIGURE 6.1: A layout of WSN based agriculture system [46]

monitors the weather information, such as temperature, dew-point, precipitation and humidity. A portable computer collects all sensory data transported over a wireless sensor network. Control commands can be transmitted to the actuator nodes from the portable controller. Water pumps and electromagnetic valves can be controlled by the actuator nodes. The end devices of wireless sensor network (WSN) are equipped with both, sensor devices and actuator. A wireless sensor network is created using a star network topology. There is a ZigBee coordinator to build and maintain the sensor network. It creates a personal area network when the coordinator starts to send the beacon packets, and allows other routers and the end devices to join the network. A liquid crystal display displays sensor data that is received by the controller. Although their designed system is a good example of the thorough implementation of a WSN based monitoring system comprising of sensor nodes and actuators, it does not cover a large area, through multi-hopping because the designed WSN is based on a star network with, base-station being a central node. The system has a lot of further issues to be addressed, as making an intelligent server, and a multi-hop sensor network.

In [47], a multi-hop WSN based precision farming system is designed to measure soil wetness, as shown in Figure 6.2. The first multi-hop chain has nodes 1 to 3. The second multi-hop chain has nodes 4 to 6. Node7 is the gateway node, not shown in Figure. The gateway node collects data sent from these nodes. In this design, a multi-hop path can

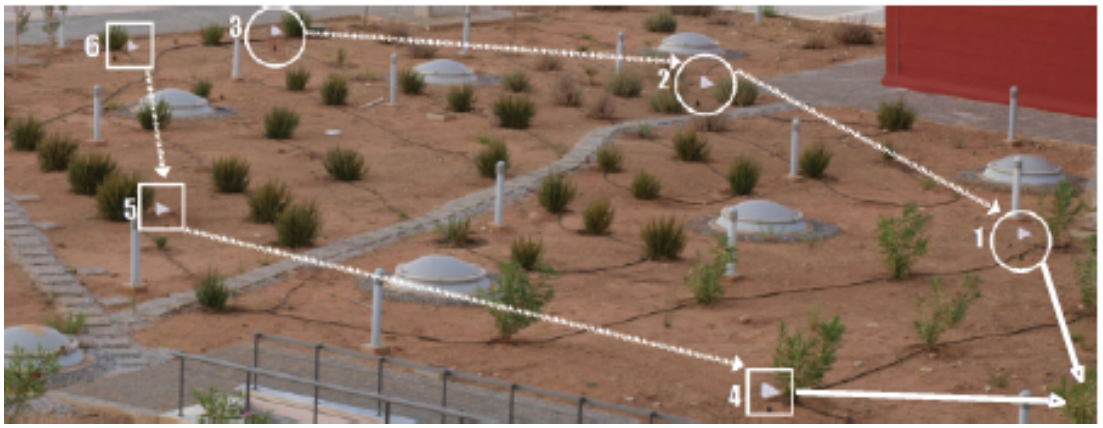

FIGURE 6.2: A multi-hop WSN based temperature and soil wetness system [47]

be established whenever a sensor node starts collecting sensory data. However, it can not provide communication if any of the intermediate nodes fails due to any reasons.

In [48], an attempt to design a reliable WSN based monitoring system is made. The designed system uses a mesh network topology. The nodes can communication with each other, and transport data between them. This enables multiple paths between each node and the gateway. However, the design does not provide an alternate pathway, when a node close to the gateway fails. The authors do not discuss on sleep scheduling operations, required to switch off the transmitter radio of the end sensor node, during the time sensed data has not changed.

Our work focuses on the methodology to design and implement a multi-hop WSN for smart agriculture system which has features like self-forming, self-protecting routes within WSN.

6.3 Comparison of Various Test-bed Setups

We can see many earlier test-bed systems; e.g. TinyOS and CrossBow motes [49] were the earlier systems for research and development works. Many empirical wireless sensor network setups were implemented in the past. For example, big scale soil water monitoring was implemented using CrossBow motes. There are also other experimental setup for environmental monitoring, such as Texas Environmental Observatory (TEO) [50] and Motelab WSN setup at Harvard University. A system developed by a well known body, named ISHMP at the University of Illinois, Urbana-Champaign [51], provides a

reliable and continuous monitoring system. There are examples of large scale monitoring, such as TinyDB [52] and SwissOM [53], but do not have analytics for integration. The system used in environmental monitoring [54] does not provide an analog interface for many sensors, which makes it incompatible with analog sensors.

OS	Min RAM	Min ROM	C Support	C++ Support	Multi-Threading Support	MCU w/o MMU	Real-time
Contiki	2kB	30kB	Partial	No	Partial	Yes	Partial
Tiny OS	1kB	4 kB	No	No	Partial	Yes	No
Linux	1MB	1MB	Yes	Yes	Yes	Yes	Partial
RIOT	1.5kB	5 kB	Yes	Yes	Yes	Yes	Yes

TABLE 6.1: Comparison of various WSN test-bed setups [49][52][53][55]

The advantages of our WSN test-bed setup can be summarized as below:

- Our WSN test-bed system consists of Arduino UNO, Raspberry Pi, XBee S2 and various sensors. It is extremely low-cost, easy to build and easy to maintain, as compared to earlier designs such as Texas Environmental Observatory (TEO).
- We are using Raspberry Pi 3 as the gateway node. It is based on customized Linux OS, which is superior to other WSN OS in terms of performance, processing, memory, programming support etc., as described in Table 6.1.
- In existing WSN test-bed setups, such as Contiki, Tiny OS etc, we need to have a separate system for a database module. We are using Raspberry Pi 3, which can provide a platform to store the data, and host web server, as it has a processor of 1.2 GHz, and a memory of 64 GB, as described in Chapter 3.
- The sensing node micro-controller hardware (i.e. Arduino) is compatible with both analog and digital interfaces. Existing WSN OS, such as Tiny OS, does not support dynamic memory allocation as it supports only nesC programming language. Arduino supports dynamic memory allocation and can be easily programmed using C and C++, which allows to use more efficient memory usage.

6.4 Related Work in Data Handling and Streaming

Many IoT based smart metering projects have used continuous streaming mode of transmission to transfer sensed data to other nodes. The number of transmissions is high in continuous streaming mode. There have been works using data aggregation method at the cluster head or the aggregator node to reduce the number of transmissions [56]. It reduces the number of transmissions to some extent, but energy is wasted in transmitting the data that contains very little or no variation. The energy consumption is still high in this case.

There have been research works [57][58] in compressing the data before it is actually sent through the network. They reduce the number of transmissions, but can not handle multiple data types; e.g. temperature and relative humidity etc. There have been some works in compressing multiple data types in WSN using entropy coding [59]. It can reduce the number of transmissions by compressing data collected over a certain period, with the entropy coding model. This in-turn saves more energy. It is good for very smaller topologies, which have not more than 3 or 4 sensor nodes. Residual values of data are calculated for an interval [-M, M], where M is the maximum range of residue interval of data that can be encoded. A probability distribution is calculated, and the code-words or tag values are generated using the entropy coding algorithm. The problem with this approach is that if the retrieved data lies outside this range, a placeholder needs to be added to maintain the ordering of data. Thus, if there are many entries which lie outside this interval of [-M, M], the complexity of the encoding algorithm increases, which makes it not a practical design choice. The receiver node needs to decode data in order to be in readable format. So, it becomes difficult to manage real-time data at the receiver when the number of end nodes increases, reporting their sensed data to the receiver node. These data compression techniques are not recommended in the constrained embedded hardware, because the processing performed by the data compression algorithm consumes more energy. We still require a simple, but efficient way of data handling for the embedded applications.

6.5 Overview of Routing Protocols in WSN

Routing protocols in WSN have be classified based on the network structure, path-establishment, protocol operation, and initiator of communication, as shown in Figure 6.3.

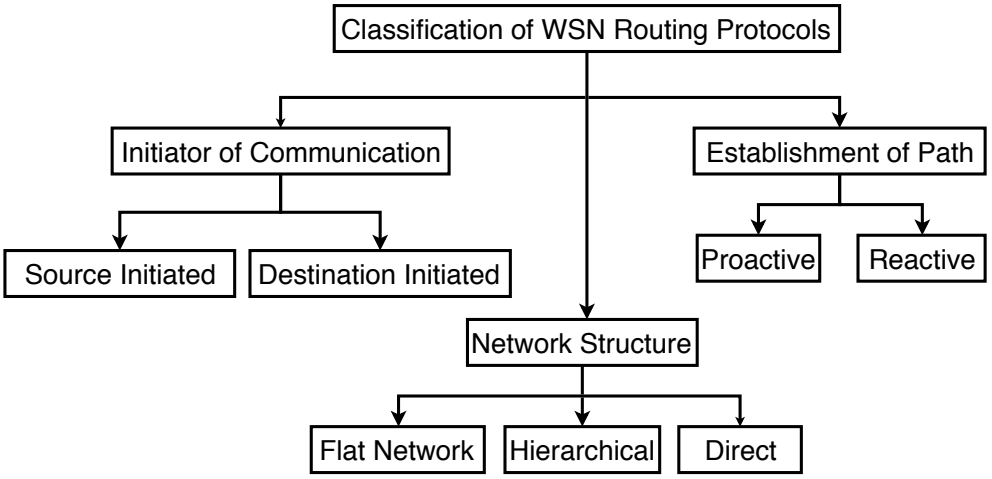

FIGURE 6.3: Classification of routing protocols in WSN [60]

In terms of path-establishment, there are different routing protocols, such as proactive, reactive, and hybrid protocols. Proactive routing protocols maintain the routing table at each node even before they require this information. It is not suitable for a large network that contains many nodes, because each node needs to keep entries for other nodes. The routing table is updated periodically when there are routes that change in the network. Reactive routing protocols search for the routes on-demand whenever a node wants to send data packet to the destination. These are also known as On-Demand routing protocols. Hybrid protocols combine features from both, the proactive and reactive protocols.

In terms of network structure, routing protocols can be classified as flat, hierarchical, and direct. In flat routing, every node has equally capability in routing. Whenever a node wants to send data, searches for a valid path to the destination. The nodes near the gateway participate more and deplete energy faster than other nodes. In hierarchical routing, a network is divided into clusters. The end nodes send their data to the cluster head, and the cluster head forwards data to the gateway. In direct routing, nodes send data to the sink directly. All the nodes are required to be close to the sink.

In terms of initiator of communication, routing protocols can be classified as either source-initiated or destination-initiated. In a source-initiated routing, all the nodes send data to the sink. Data can be sent using time driven or event-triggered reporting approach. In destination-initiated routing, the sink node sends query packets to other nodes. This introduces many overheads because many requests are flooded through the network.

6.5.1 Flat Network Routing Protocol

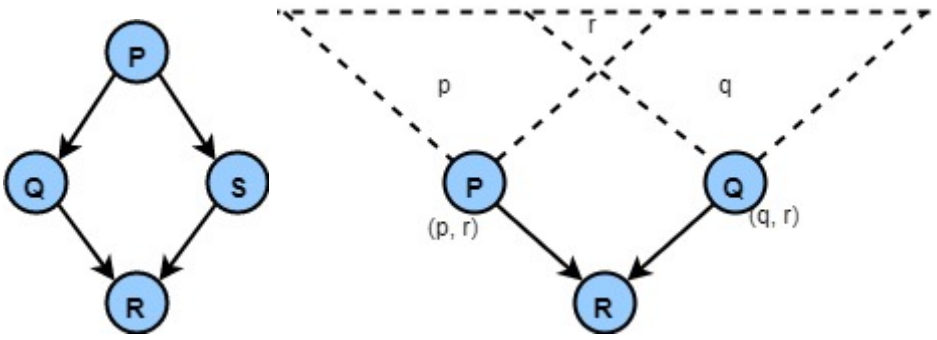

FIGURE 6.4: a. Implosion b. Overlap

Flooding and Gossiping: Flooding and Gossiping protocols [60] are applicable to flat routing network architecture. In flooding, each node broadcasts the data packet to every other node that is in direct communication with. Each of these nodes forwards the data packet by flooding as well. Figure 6.4 (a) shows a scenario where implosion can occur. The sink node might receive multiple copies of the same packet. Node P starts by flooding its data to all of its neighbors. R gets two copies of data which is not necessary. The flooding protocol exhibits many disadvantages, such as overlap, implosion and blindness of resources. Overlap happens when two nodes which are in the communication range, send similar message packets to the same node, and implosion occurs when two nodes send duplicate data packets to the same neighbor. Figure 6.4 (b) shows different regions, p and q, and an overlapping region r, and shows a scenario where overlap can occur. Two nodes (e.g. P and Q) cover an overlapping geographical region r. R gets the same copy of these data from these nodes. Resource blindness is caused because energy constraints are not taken into consideration in routing. This can lead to fast depletion of energy resources. Implosion can be avoided using the gossiping technique. It avoids implosion by randomly selecting the next hop, and sending data packet only to that address. The method is characterized by high latency in the delivery

of the data packet to the destination. Hence, these protocols are not suitable for energy constrained devices.

Sensor Routing Protocols for Information via Negotiation (SPIN): In SPIN [61], every node maintains a routing table about its direct neighbors. There are three stage in SPIN, as shown in Figure 6.5. There are 3 types of messages defined in SPIN; ADV, REQ, and DATA. ADV is used for advertising new data packet, REQ for requesting data, and DATA is the actual data payload itself. In the first stage, each node sends ADV containing meta-data information to all its direct neighbors, as shown in Figure 6.5 (a). The format of meta-data is application specific, and is not mentioned in SPIN. For example, nodes might specify their IDs to carry meta-data if they are located at a certain area. In the second stage, nodes, which are interested in advertised data respond by sending REQ message, as shown in Figure 6.5 (b). In the third stage, the initiator node sends data to those interested nodes, as shown in Figure 6.5(c).

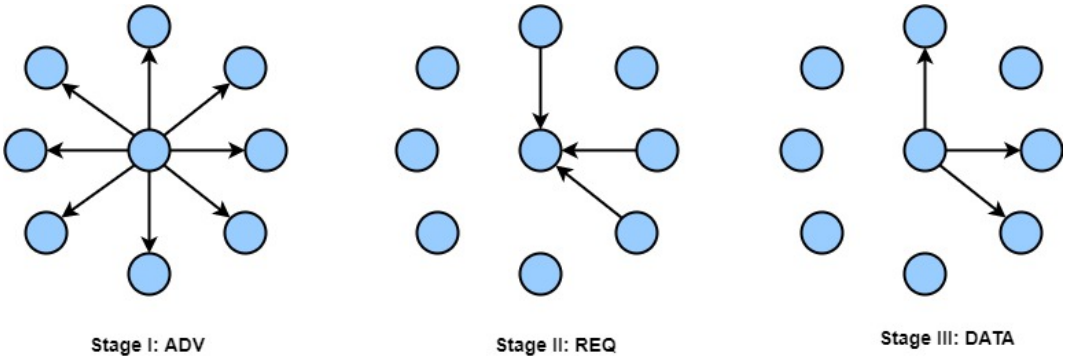

FIGURE 6.5: The three stages SPIN goes through [61]

Direct Diffusion: Direct Diffusion [62] is a reactive protocol suitable for flat-networks. It applies data-centric routing, where queries are created in particular areas.

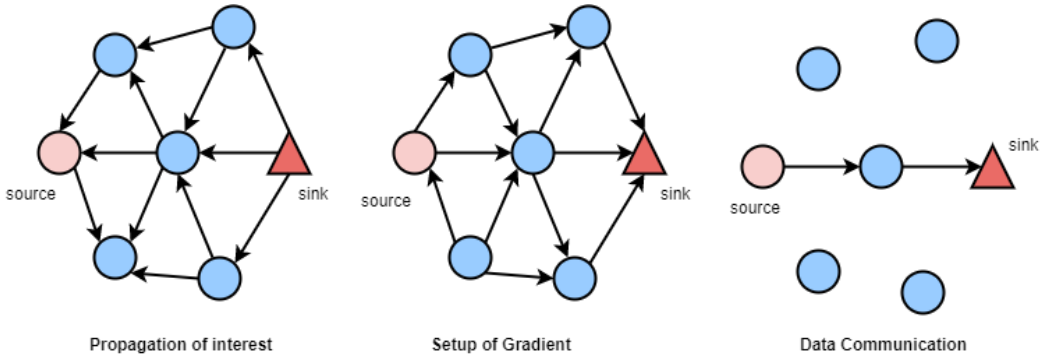

FIGURE 6.6: a. Interest propagation b. Gradient setup c. Data communication [62]

There are three distinct stages involved in this routing; propagation of interest, setup for gradient, and data communication, as shown in Figure 6.6. During the interest propagation stage, the sink node floods an interest packet through the network, as shown in Figure 6.6 (a). The interest packet contains attribute and value pairs to describe a task. For example, a list of attribute and value pairs that describe an animal tracking task can be expressed as below:

```
type:  four-legged animal,
time interval:  25 ms - send events every 25 ms,
time duration:  15 sec - for the duration of next 15 seconds,
rect:  [-50, 50, 100, 200] - within this area from the sensors
```

This prevents from receiving undesired data. The interest packet contains the rate at which the sensor nodes need to transmit data to the sink. The nodes maintain the cache of "interest". It contains the entry containing the data rate for each neighbor address. Figure 6.6 (b) shows the second stage i.e. gradient setup. In this stage, nodes verify whether the attribute value pairs are matching or not, and start transmitting data for the matching pairs, at the data rate mentioned in the entry table, to all the addresses, which propagated the interest packet. A fixed data rate is set, at which data should be sent from the nodes. This is called gradient setup. One or more paths will then be enforced by the sink by sending another interest packet. Figure 6.7 (c) shows the data communication. In this stage, nodes start sending their data at the rate specified in the message packet of reinforcement process. The disadvantage in diffusion is that it causes more number of transmissions during query and reply processes. This results in high energy consumption. In addition, a sensor node needs to have a larger memory available to store data of interest.

Rumor Routing: Rumor routing [63] is a destination initiated protocol. It uses flat network architecture and hybrid network routing. It is a derivative of direct diffusion routing, but events are flooded rather than interests. Each node maintains a table of events and neighbors. Whenever a certain event is sensed by a node, it generates an agent with certain Time-to-Live (TTL) value, has it on a random path. An event is added to the event table. All visited nodes form a gradient setup to the event. A query is generated by the sink whenever it requires an event. A node which knows the route to that event responds by looking its event table, and returns it through the source to

destination path. Rumor routing does not require to flood the query packets through a network, and works very well when the number of events is less. Whenever the number of events is very large, the cost of maintaining an event table is very high. It is not suitable for a network that contains the large number of events. It depletes the energy quickly because of the high number of transmissions.

Minimum Cost Forwarding Algorithm: Minimum Cost Forwarding Algorithm [64] is a source initiated protocol. It uses flat network architecture and proactive routing. In this algorithm, each node maintains a least cost value to the sink node. Initially, all nodes set the minimum cost value to infinity, and the sink node sets the minimum cost value to 0. The sink node starts broadcasting the packet with its minimum cost value. Each neighbor node which receives the packet updates its cost value (i.e. hops counts, delay) if the cost of received packet, including path cost is lower than the current cost value, and broadcasts if it is the least cost path. By doing so, the data packet which is generated from a node travels via a least cost path. The disadvantage of using this algorithm is that it needs to periodically update the cost value.

Gradient Based Routing: Gradient Based Routing [65] is a destination initiated protocol. It uses flat network structure and reactive routing. This is also a derivative of direct diffusion. In this routing, the interest packet also includes the cost path (i.e. number of hops count), when it is propagated through the network. This way, each node calculates its height based on minimum number of hops to reach to the sink node, which forms the gradient setup. If there are nodes having the same number of hops count, the node elects the next hop based on various election methods. Next hop is either randomly chosen or based on checking the energy level threshold value of a node. It increases a height if the energy level of node fall below a certain threshold. A node shall avoid sending data to neighbors if it finds those nodes are on different stream of messages. Gradient Based Routing has the disadvantages of introducing flooding packets of interest as in direct diffusion.

Energy Aware Routing: Energy Aware Routing [66] is a destination initiated protocol, like the direct diffusion. It uses flat network topology and proactive routing. It maintains multiple routes at each node. Whenever a node wants to send data, a path is randomly chosen according to its probability profile. The total energy cost is calculated at each node. The probability is calculated based on the energy metric. This protocol has three

stages; setup, data transmission, and path maintenance. In the setup phase, packets are flooded from the sink node and propagate through the network maintaining the routing table at each node. The total energy cost at each node is calculated. The next hop is selected on the basis of proximity to the destination. In the second stage, each node sends data to the next hop by randomly selecting a neighbor node from its routing table. During the last stage, each node is periodically updated regarding route changes through localized flooding, and the path is maintained. The disadvantage of this protocol is that every time the same path is used to send the packets, there is a high probability due to energy depletion of the intermediate nodes.

6.5.2 Hierarchical Network Routing Protocol

Low Energy Adaptive Clustering Hierarchy (LEACH): LEACH [67] uses hierarchical structure, as shown in Figure 6.7. It uses two stages; setup and steady. During the setup stage, clusters are created and the cluster heads are selected. A random number between 0 and 1 is chosen by every node, and a node becomes a cluster head if the selected number is below the certain threshold value. The threshold value is shown in Equation 6.1.

$$T(n) = \begin{cases} \frac{p}{1-p*r*mod\frac{1}{p}}, & \text{for } n \epsilon G \\ 0, & otherwise \end{cases} \tag{6.1}$$

where p is the probability to become cluster head, r is the current round, n is the given node, G is the set of nodes that have not become cluster head yet.

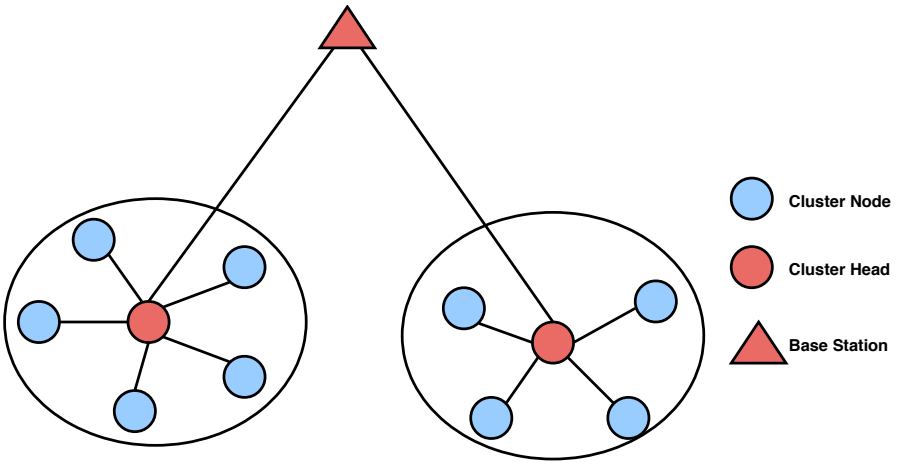

FIGURE 6.7: Low Energy Adaptive Clustering Hierarchy (LEACH) [67]

Cluster head informs all its cluster nodes that it has become cluster head. Once a node has become cluster head, it can not become cluster head again unless all its cluster nodes have become cluster head at least once. The end nodes can save energy by going into low power save mode during the time they are not sending data. Cluster nodes send data to cluster head based on schedule TDMA transmission. Data is sent to the sink node during the second stage i.e. steady stage. The cluster head aggregates the data.

Disadvantages of LEACH are as below:

- When a cluster head dies due to any reason, the cluster members do not find routes to the sink node. There can be an uneven distribution of clusters based on the number of nodes, which can increase the energy consumption.

Hierarchical Energy Aware Routing: HEAR [68] is based on data aggregation. It assumes that the cluster head has better transceiver and energy as compared to cluster nodes. Cluster head broadcasts a message periodically, and a receiving node decides whether it needs to update its routing table or not. Beacon is transmitted only to the cluster members. If better link is available, the table is updated. The disadvantage of this routing is that it requires cluster head to be equipped with better transceiver.

Threshold Sensitive Energy Efficient Sensor Network Protocol (TEEN): Threshold Sensitive Energy Efficient Sensor Network Protocol (TEEN) [69] is hierarchical based routing protocol. It uses reactive routing protocol. The nearby nodes form the clusters. The cluster head specifies threshold values (i.e. high and soft threshold values) to its cluster members. Hard threshold is the minimum possible value of the sensed attribute. Soft threshold is the small change in sensed attribute, which switches the transmitter radio on.

6.5.3 Comparison Between Flat and Hierarchical Network Routing Protocol

Hierarchical Network Routing Protocols:

- Hierarchical network protocols have cluster heads with higher capacity, which can localize the energy consumption as all its cluster members in a cluster report only to

the cluster head. Data aggregation at the cluster head is performed, which reduces the number of transmissions, and in-turn helps to reduce energy consumption to some extent.

- Scheduling is done based on reservation.
- Routing is non-optimal.
- Cluster heads expect to have better transceiver and hardware.
- Complex algorithms are implemented to select the cluster head, and they require to rotate the cluster head periodically based on the probability factor.
- Cluster heads consume more energy than any other nodes, and the network can become disconnected if the cluster head is not rotated regularly within a cluster (as described in LEACH).

Flat Network Routing Protocols:

- Flat network routing protocols are scalable, as all the nodes participate uniformly in network routing tasks, and a node can send data to its direct next hop neighbor only.
- Scheduling is done based on contention.
- Optimal routing can be done.
- All nodes have equal capabilities in terms of radio and hardware.
- Simple procedures are implemented to choose the next neighbor.
- Hot-spot problems are created as the nodes near the sink node participate more and deplete energy faster than other nodes.

6.6 Problems in WSN Routing

We can summarize that both, the flat and hierarchical network topology have advantages, as well as disadvantages. In flat network topology, the nodes near to the sink node participate more, and deplete energy faster than other nodes. Flat networks have many

limitations, such as flooding the packets of interest, as seen in direct diffusion. In the hierarchical structure, the cluster head requires to be equipped with a better transceiver than other nodes, like in LEACH, and complex algorithm is executed while choosing a cluster head. There is a problem of reliability in routing because if one of the parent nodes fails, the child nodes do not find route to transport data to the sink.

It can be seen that not many routing protocols, as briefly described above consider the number of hops count, as one of the parameters in their routing algorithm, except Gradient-Based Routing (GBR) and MCFA, as explained above. They do not consider other parameters in the routing, such as better link quality, if the nodes have the same hops count, and hence can not guarantee the maximum success rate of the packet reception. The disadvantage of using a single parameter while choosing the next hop is that, every time the same path is used to send the packets, and a high probability of network failure occurs because of the depletion of energy of the nodes along that path. They also do not guarantee the confirmed delivery of packets in case of occasional failure of one of the intermediate nodes in the network.

6.7 Link Quality Estimators in WSN

Link quality depends on many factors, such as environmental factors, interference, dynamic network conditions, radio pattern etc. The basic building block of routing in wireless sensor networks is the accurate estimation of link quality [70].

Link Quality Estimators (LQE's) can be classified into two major categories. They are hardware estimators and software estimators [70]. In hardware estimators, LQE can be extracted directly from the radio module. The examples of hardware estimator are Received Signal Strength Indicator (RSSI), Signal-to-Noise Ratio (SNR) etc. They are calculated based on successfully received packets only. Therefore, when the link undergoes through a major packets loss, transmission performance is overestimated. In software estimators, the packet reception ratio (PRR) is calculated based on the average number of successfully received packets to the average number of transmitted packets. The examples of software estimators are Packet Reception Ratio (PRR), Required Number of Packet Reception (RNP), Acquired Reception Ratio (ARR) etc [70]. They can be further classified, based on PRR or re-transmission based LQE's. The

PRR based estimators depend on the calculation of the packet reception ratio (PRR) metric. The examples of PRR based-estimators are four-bit, ETX, WMEWMA. The re-transmission-based Link Quality Estimators (LQE's) depend on the RNP metric.

Most of the link quality estimator algorithms, such as PRR and RNP compute the link quality in one direction. ETX is more efficient as compared to other software estimators. ETX, based on PRR uses bidirectional links (i.e. PRR_f as forward link, and PRR_r as opposite link) while computing the link quality, but does not consider the highest link quality of one of the links [70].

For a link, the ETX metric can be calculated using a formula, as shown in Equation 6.2.

$$ETX(w) = \frac{1}{PRR_f \times PRR_r} \tag{6.2}$$

where,
PRR_f = The packet reception ratio in the forward direction.
PRR_r = The packet reception ratio in the opposite direction.

Let's say there are two routes between nodes 1 and 2. For first route, the link quality from node 1 to 2 is 20 in both directions. For second route, the link quality from node 1 to 2 is 45, from node 2 to 1 is 1. With ETX, first route is selected in routing decision because ETX does not consider the highest link quality of one of the links. We observed that with the same distance between the nodes and same environmental condition, the forward and reverse link qualities are different. This can occur due to random collisions of the packets. In our design, we consider the highest link quality $max(PRR_f, PRR_r)$ from the forward and reverse links.

6.8 Conclusion

We briefly reviewed the related research areas in smart agriculture system, a variety of experimental setups, data streaming modes, and problems in WSN routing. While designing a sensor network that can be used to transport data, we need to carefully decide which features of existing protocols can be adopted, and which should be further improved. We have addressed some of these issues. We will discuss them in Chapter 7.

Chapter 7

Proposed System

In this chapter, we focus on the detailed system design choices, programming methodology of sensor nodes, including our implementation of new routing algorithm in WSN using different metrics, such as hops count, packet reception ratio, battery voltage to obtain more reliable, scalable and efficient routing. We will also describe our proposed methods used to reduce the number of transmissions in WSN routing, and the methods adopted to save energy consumption.

7.1 Proposed Design Choices

The system consists of physical layer, gateway layer, middle-ware layer, and application layer. The design requires to extract and modify many features at each layer, as shown in Figure 7.1. They are described below. The system is designed having in mind the need for routing efficiency, reliability, scalability, and minimization of the number of transmissions occurring in the WSN.

7.1.1 Physical Layer Design Choices

The physical layer consists of sensor devices, micro-controller unit (MCU) and the XBees. The physical layer is designed to be the scalable, less complex, has low computing cost, maintains low-energy consumption and be cost-effective to save in both, energy and money.

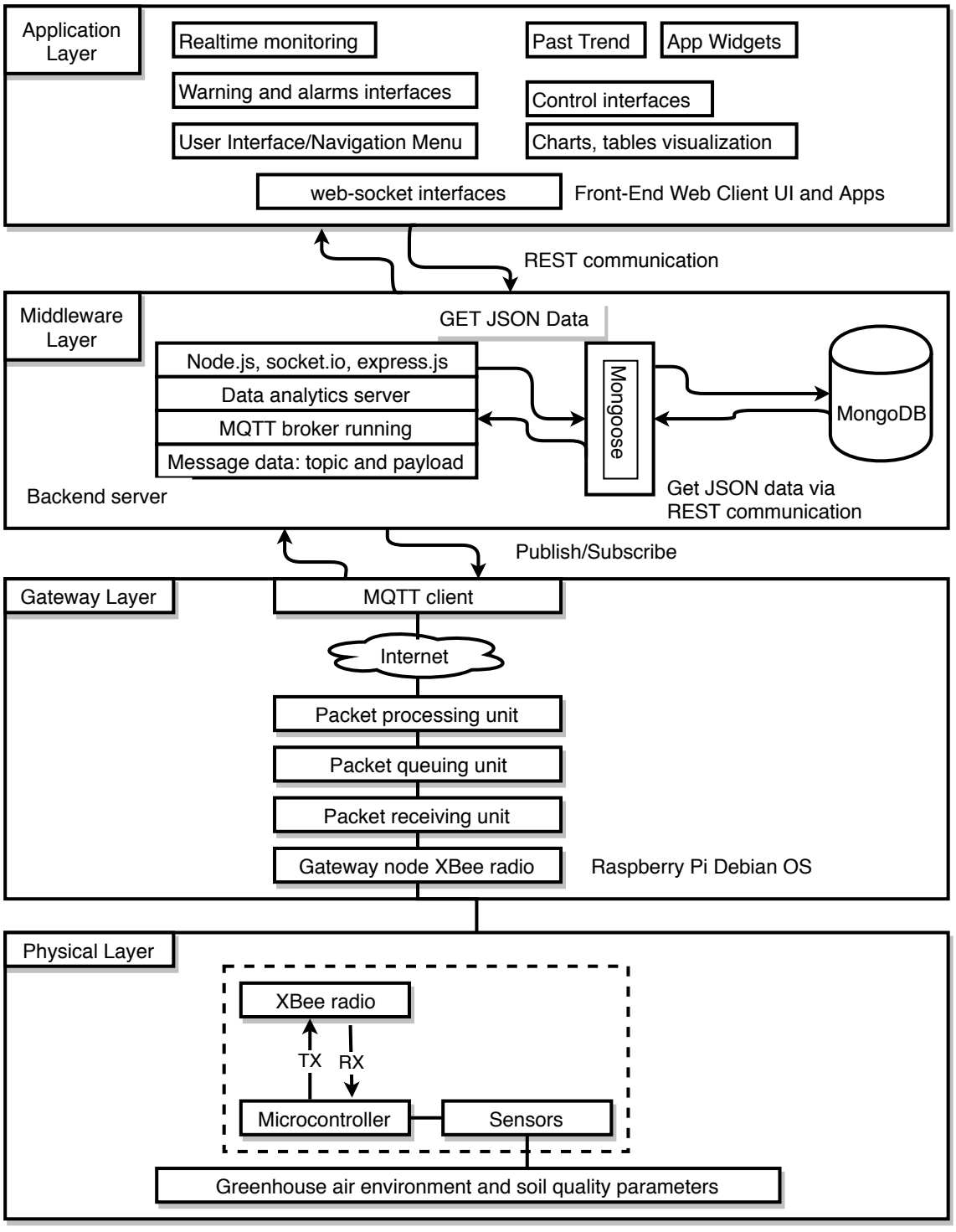

FIGURE 7.1: Description of proposed design

The following design criteria are taken into consideration while designing the physical layer.

- The way sensor nodes are deployed affects the efficiency in sensor networks. Our design accounts for the fact that several sensors to be deployed in the greenhouse area have to be located far from the gateway node. Therefore, sensor nodes are deployed so as to cover entirety of the greenhouse.
- New sensor nodes can be added "on-the-fly" to an existing network deployment, and is done so by simply configuring XBee using the XCTU software. The application code can be easily programmed using Arduino IDE.
- The complexity of micro-controller programming is kept low to reduce the energy consumption.
- No special hardware (e.g. GPS, RFID, high power transmitters etc.) is used in the sensor node, which makes it simple and practical.

7.1.2 Data Handling Choices

- Data Delivery Methods: We have adopted a source-initiated event based data transmission method. Objective is to reduce potential flooding of packets. We observed that event triggering is the better choice in terms of saving energy. A proposed threshold based event triggering method is explained in Section 7.5.
- Data Aggregation: Within typical greenhouse environments, certain degree of noticeable temperature variability should be expected for distances in the range of or larger than 20m to 30m. Thus, to increase accuracy, we should have several sensors located close to each other with distance less than 20 to 30m, and then take the average of the measurements. If we do not handle data efficiently, this will increase the number of transmissions of packets from the nodes to the sink, and consumes more energy from the nodes. A new algorithm based on data aggregation is proposed. The detailed explanation of this algorithm is given in Section 7.4.

7.1.3 Improvement in WSN Routing

We have implemented improvements on existing WSN routing protocols using an efficient cost metric for making routing decisions within WSN, and introducing reliability. We describe them below.

7.1.3.1 Cost Metrics in WSN Routing

Next hop address is chosen based on the lowest number of hops count as primary parameter in deciding the routing path to the sink node, however, other criteria are also taken into consideration, e.g packet reception ratio, current battery voltage. We noticed that not all nodes that are in communication range or have been discovered during the neighbor discovery process gave equal number of transmit status responses i.e. very few responses packets were received for some sensor nodes because of poor link quality. Many factors affect the link quality, such as interference, distance etc. Thus, it is important to consider the link quality in order to minimize the packet failure rate. A simple formula is introduced and given in Equation 7.1 and 7.2. The description of how the neighbor discovery process works is explained in Section 7.2.1.

$$LQE = PRR \times (node_{battvolt} - critical_{battvolt}) \tag{7.1}$$

$$PRR = \frac{Total\ no.\ of\ acknowledgment\ packets\ received}{Total\ no.\ of\ packets\ sent} \tag{7.2}$$

where LQE is the link quality estimator, PRR is the packet reception ratio, and is computed at both sender and receiver sides, $node_{battvolt}$ is the battery voltage of the node, and $critical_{battvolt}$ is the critical battery voltage of the node.

Unlike other PRR based bidirectional link quality estimators, such as ETX [70], in our design, each node keeps the highest value of the packet reception ratio (i.e. $max(PRR_f, PRR_r)$) from the forward and reverse links. PRR_f is computed at sender, and PRR_r is computed at the receiver but communicated to the sender via response packet of LQE. Each node takes a highest value because forward and reverse PRR values are different due to many reasons, such as random collisions of the packets, environmental factors

etc. If we simply keep the average value from both the links, the estimation might not be accurate. The total number of transmissions is set to 20, and the minimum number of acknowledgments is set to 5. Based on our observation, when a node received at least 5 ACK packets, it was able to deliver the packet with minimum retries. Therefore, these parameters are set keeping in mind that a node can send or receive packets without much loss.

According to battery discharge graph [71], 70 % of battery used is considered as critical battery level. The battery level is measured by reading the voltage from the battery voltage sensor [72]. It can be seen that battery voltage reading is not linear related to its discharge rate. The battery which we used during initial testing (i.e. 9 V Alkaline battery), has 6.5 V output voltage when it is 70 % discharged. When the battery level of the node started reaching to the critical stage, the node went through a major packet loss but was able to reach to the destination node with less than 20 retry counts. So, in this stage, a node starts sending link quality estimation packets, and the current battery voltage is communicated to other nodes in the response packet of link quality estimation. If the LQE value does not satisfy this requirement (i.e. if current battery voltage of the neighbor node reaches the critical battery voltage value, or the total count of ACK packets during PRR calculation is less than 5), the sensor node will not be considered for acting as the next hop. If no neighbor meets this criteria, it sends request to read LQE again. A node still does not consider the node which reaches the critical battery voltage even though a node receives the acknowledgment packets of more than 5. So, our LQE estimation formula is multiplicative of both PRR and the difference between the current battery voltage minus the critical battery voltage of the node. It is kept to avoid a major packet loss, which can occur due to the critical battery level of the node.

7.1.3.2 Reliability

The designed system provides alternative route in case of occasional failure of intermediate nodes. A node tries to send packet to the best neighbor first which has the lowest hops count and good link quality. If the node does not acknowledge for a certain retry counts, it will consider another best neighbor from the routing table. The detailed explanation of our proposed routing algorithm is explained in Section 7.2.1.

7.1.3.3 Scalability

A node can join a network dynamically and gets integrated into the network automatically with no or minimal changes to the existing network. Whenever a new node wants to be added in the existing network, it acquires the number of hops count, and the current battery voltage level from the nearest neighbor in order to update its routing table. It then establishes multiple paths when data is sent from source to reach the gateway node. A method described in Section 7.2.1 provides the detailed procedure.

7.1.4 Middleware Layer Protocol Design Choice

We have chosen MQTT, as the protocol for the communication between the gateway node and the server. This way we can achieve interoperability between the gateway and the application layer. We have used python package for mqtt [73]. The application layer receives payload with subscribed topic and payload in JSON format.

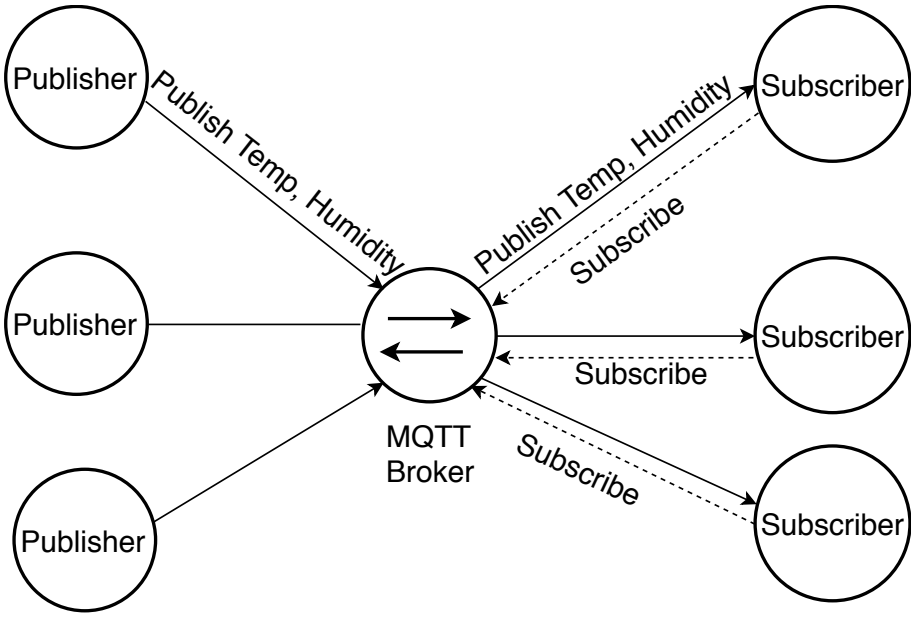

FIGURE 7.2: MQTT protocol [74]

MQTT [74] is a light-weight messaging protocol used in IoT applications, mainly designed for constrained devices, such as embedded devices with limited processor and memory resources. It is built on top of TCP/IP. A set of events or messages are exchanged between the source and the destination, based on topics of interest or desire. Figure 7.2 shows, there are three components in this protocol; publisher, subscriber, and

broker. Publisher is a component that generates the messages, which contain the actual payload, while one or many subscribers consume the messages. Each message generated by the publisher must contain a topic, and the subscriber needs to be subscribed to that. Broker is a component that interacts with both the publisher and the subscriber. Publisher generates data and subscriber consumes it by connecting and communicating with the broker. Until both publisher and subscriber connect to broker, they can not talk each other. Broker is identified by each node based on IP address or host name, and a port number assigned.

Three QoS levels are defined by MQTTv3.1. They are mentioned in Table 7.1. We used QoS (2) while publishing sensor data.

QoS	Message Delivery	Semantics of Delivery	Guarantee of Delivery
0	≤ 1	At most once	Best efforts; no guarantees
1	≥ 1	At least once	Guaranteed delivery; Possibility of duplicate packets
2	$\equiv$	Exactly once	Guaranteed delivery No duplicate packets

TABLE 7.1: QoS levels of MQTT [74]

7.1.5 The Backend and Application Layer System Design

The backend layer is designed using the MEAN stack based web server, which consists of MongoDB, Express.js, AngularJS and Node.js. The backend layer communicates with MonogoDB database service using REST communication. MongoDB which offers many features like data aggregation model using the concepts of data processing pipeline called aggregation pipeline framework. We have discussed about the advantages of various modules of our backend and application layer in Chapter 4.

7.2 Programming of Sensor Node

The basic process of node initialization is shown in Figure 7.3. The process of main loop of sensor nodes is shown in Figure 7.4. The XBee library provided for Arduino [75] and python package [76] for Raspberry Pi are used for network scan operation.

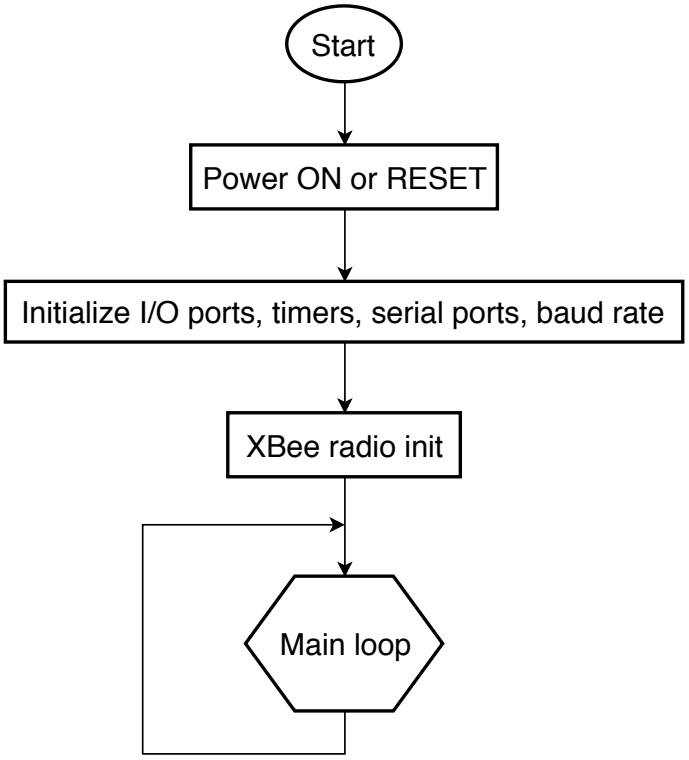

FIGURE 7.3: Initialization of sensor node

In the initialization phase, all pins of the micro-controller unit, serial ports of the development board, baud rate get enabled. After the micro-controller gets initialized, we can see Tx/Rx and LED pins "ON". It shows that the XBee radio is initialized. The process is now under the main loop of the controller.

The main loop is slightly different for intermediate nodes (i.e. full-function devices) and end sensor nodes (i.e. reduced-function devices). As shown in Figure 7.4, each sensor node (i.e. intermediate node) first starts sending LQE packets to the discovered nodes after the neighbors are discovered using the neighbor discovery services. The nodes will not be added if they can not satisfy the LQE threshold set by the node as described in Section 7.1.2. Each node first initializes the `self_hop_count` value to 50, as maximum value. When the sensor node receives a network setup message, which contains the source address, the destination address, current battery voltage, and the number of hops count value, it forwards the messages to all its neighbors. Before that, each node maintains the `self_hop_count` after it receives `number_of_hops_count` from its neighbor sensor nodes. If `self_hop_count` is greater than `number_of_hops_count`, it will update `self_hop_count` to `number_of_hops_count` plus 1.

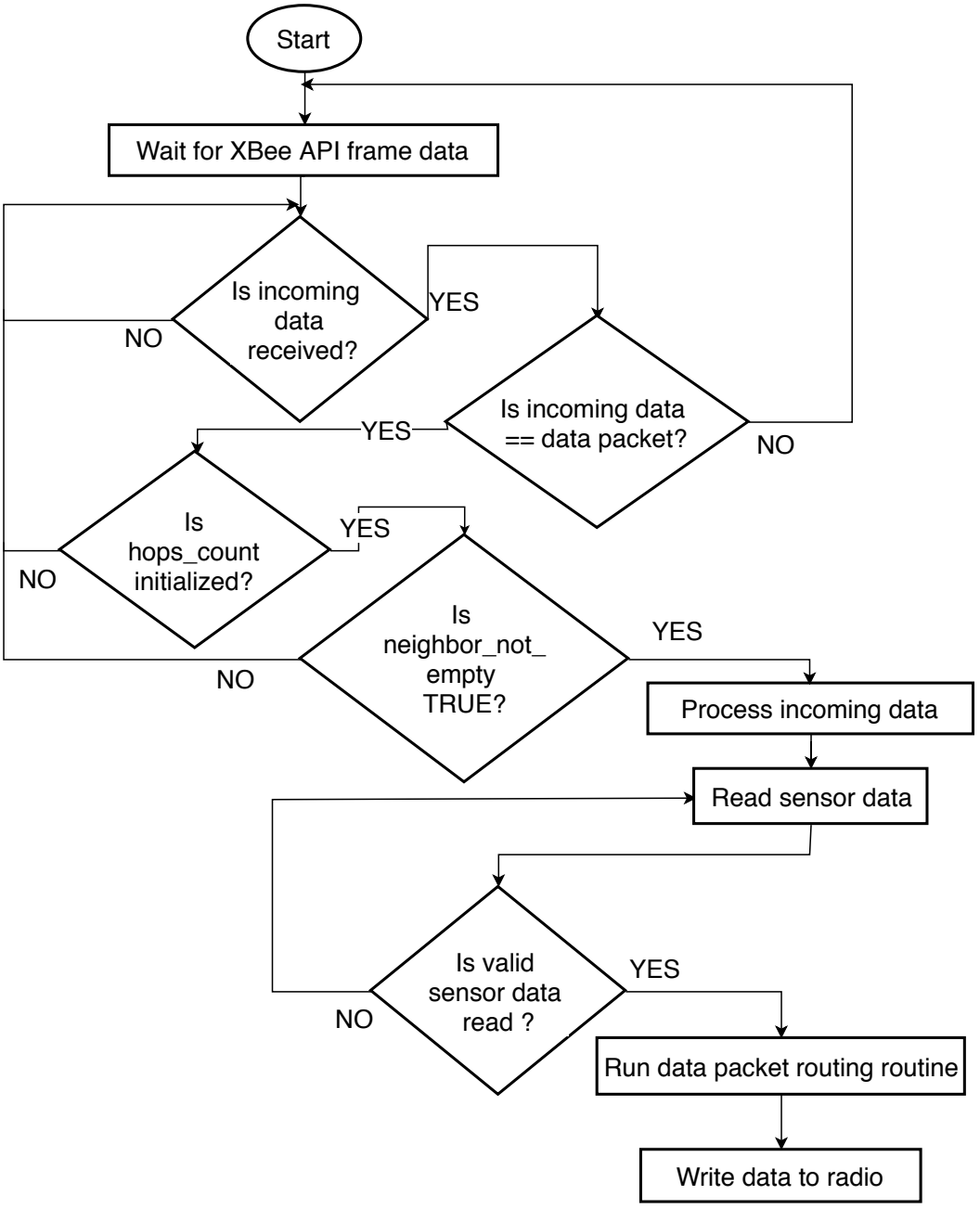

FIGURE 7.4: Main loop of sensor node

In the routing phase, it checks the neighbor array to see if it has any neighbor. It reads sensor values (e.g. temperature and humidity) from the sensors which are connected to it. A call to aggregation subroutine, `sync_and_perform_aggregation` is made as mentioned in Section 7.4, and call to `send_data_to_destination` routine is made to send data to the next hop address which passes the source address, and the payload of the data. The XBee 64-bit addressing is used and the next hop destination address is selected from the neighbor array. The detailed explanation of proposed routing algorithm is provided in Section 7.2.1. The main loop of the end sensor nodes (i.e. reduced-function

device) is simpler because they do not perform the neighbor discovery services. Such node maintains its routing table after it receives the network setup message or LQE message from its neighbor nodes, which contains the source address, the number of hops count, and the current battery voltage level.

7.2.1 Proposed Algorithm for Data Routing

XBee is configured in API mode via XCTU because API mode facilitates to change the destination address from the host side. Before routing sensor data, the network will go through the neighbor discovery, the LQE, and the network setup. They are explained below. This process is stopped after each node receives network setup message and has initialized the necessary parameters, such as the number of hops count, battery voltage, and the packet reception ratio.

7.2.1.1 Neighbor Discovery and LQE

The neighbor table is maintained by calling `network_scan` function, which calls neighbour discovery services. The XBee explicit transmit request and response frames are used while performing this operation. The discovered nodes are stored in the `nodes_info` structure array. Sensor nodes now start sending LQE packets, and maintain the link quality about its neighbors in `nodes_info` structure. The discovery and link quality determination process can be explained in Figure 7.5.

7.2.1.2 Network Setup

The procedure of network setup process is mentioned below. The network setup process is shown in Figure 7.6. For simplicity, only the best paths (i.e. less number of hops count and good link paths) are shown.

- Each node waits for the network setup message from the gateway node before it starts sending data.
- The gateway node sends the network setup message to all the neighboring nodes that are discovered after scanning the network. The scan operation also discovers a self node which is not used for forwarding the setup packet.

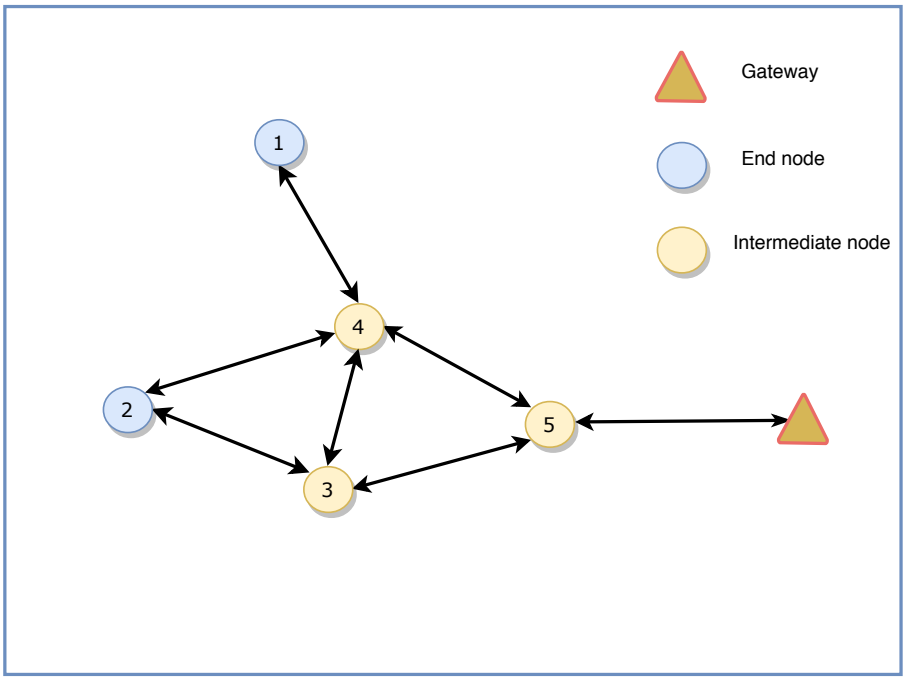

FIGURE 7.5: Neighbor discovery and LQE process

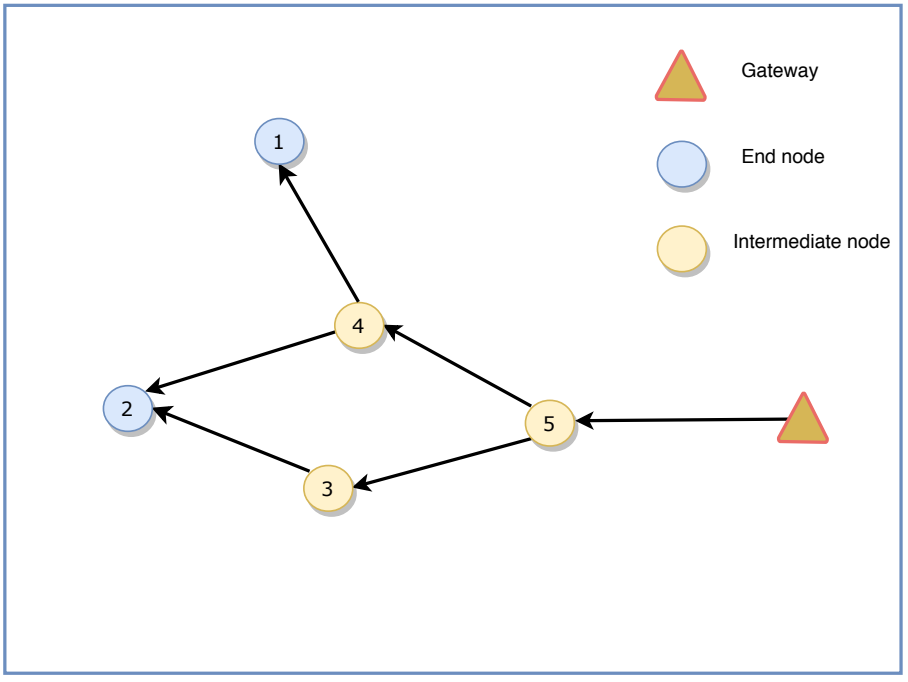

FIGURE 7.6: Network setup process

- When the sensor node receives a network setup message, it forwards it to all its neighbors. Before that each node first update the message with the source address, with the self address, and maintains the `self_hop_count` after it receives `number_of_hops_count` from neighbor nodes. If the `self_hop_count` is greater than `number_of_hops_count`, `self_hop_count` is set to `number_of_hops_count` plus 1.

- The network setup message contains 64-bit address of the source and the destination, and the number of hops count. The retry count value is kept 10 because a node sometimes required more than few retries to reach setup packet to another node. Based on our observation, if we keep retry counts less than 10, a network setup packet might not reach to the neighbor nodes, and the very high value will increase the network setup time. The network setup packet is not forwarded until it acknowledges to the source node who sent it.

- `number_of_hops_count` of a source node also gets updated when a node receives a setup packet.

- The end sensor node periodically wakes up from sleep and requests from its neighbors in an hourly basis to get the updated values of the number of hops count, current battery voltage etc. The neighbor nodes update these parameters.

- If the node is powered on after the network has already setup, it initializes the number of hops count, the battery voltage from the response packet of LQE.

7.2.1.3 Sensor Data Routing

The flow of routing algorithm is described according to Figure 7.7. A sensor node does not start sending data unless the LQE process is completed and the network setup is received from its neighbors or hops count is initialized from its neighbor. If there are no neighbors, `is_no_neighbor` flag is set at each node. The data routing routine starts first by checking `is_no_neighbor` flag is set or not. It reads the `nodes_info` array, which contains the neighbor addresses, and checks for suitable the next hop address. The decision to choose next hop address is done based on number of hops count as primary parameter. Only the neighbors which have lower number of hops than `self_hop_count`, will be considered. The 64-bit address of an original sender node is not considered to

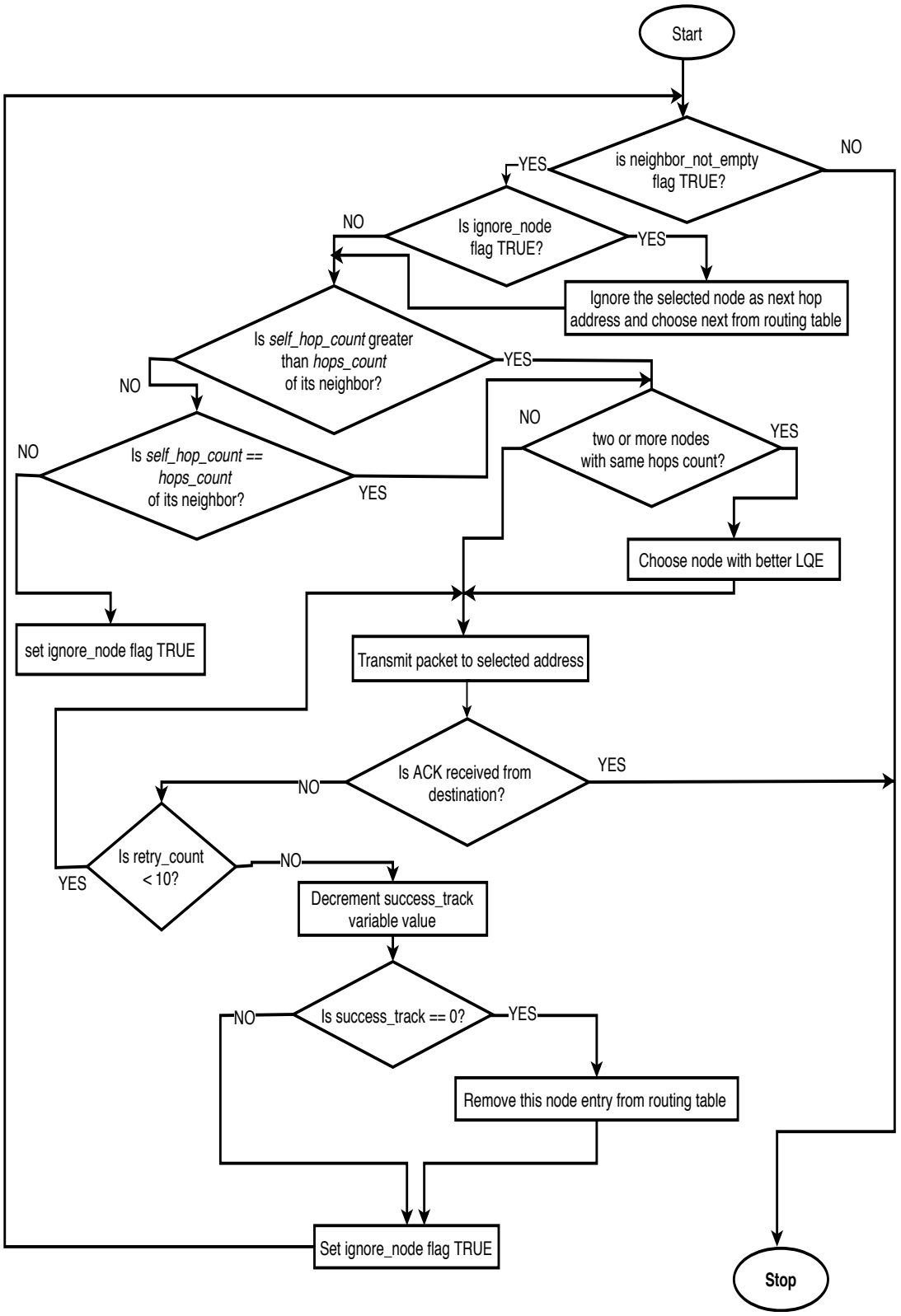

FIGURE 7.7: Flowchart of routing and failure handling

route the data packet by intermediate nodes in order to prevent sending data packet back to the sender node. It will then check for the neighbor node with less number of hops count. If two neighbor nodes have same number of hops count, then LQE value is checked. If it finds the same values of LQE, it will randomly choose one. If the `ignore_node` flag is true, then it ignores this selected node address and tries to find the next best neighbor. After the destination address is elected, `send_data_to_destination` routine is called which contains source address of a node sending data, destination address of the next hop using 64-bit addressing mode and the payload (i.e. sensor data). The data packet is created and transmitted to this node address using XBee transmit request API frame. The node then waits for a maximum of 200 ms for an acknowledgement (i.e. XBee transmit status response) from the destination node and it re-transmits again if it does not receive an acknowledgment. When it receives an acknowledgment, the node tries to send the next packet. The maximum number of re-transmission counts is configured to 10. Based on our observation, a node which is busy in transmitting a packet required a higher number of re-transmission counts to receive the packet while another node tries to send the packet to it at the same time. If this value is kept less than 10, the packets can get lost because some links were less reliable, and the link quality are extremely dynamic due many factors, such as interference, obstacles caused by walls, trees etc. If maximum number of re-transmission counts is exceeded, it sets the `ignore_node` flag for this node and selects the next best neighbor from the list. When retry counts exceed value greater than 10, it also decrements `success_track` variable value for that node. If `success_track` variable value for this node is zero, it deletes this node from neighbor list array so that in the next cycle node does not try to send data to this node. The `success_track` variable value is initially configured as 99. Node again tries to send data to previously ignored node once in 20 iterations to check if it has repaired or not before it gets deleted. This is done because we observed that sometimes nodes started to receive the packets afterwards either because it detects less collisions or medium is freed, even though we marked them as `ignore_node` flag during routing.

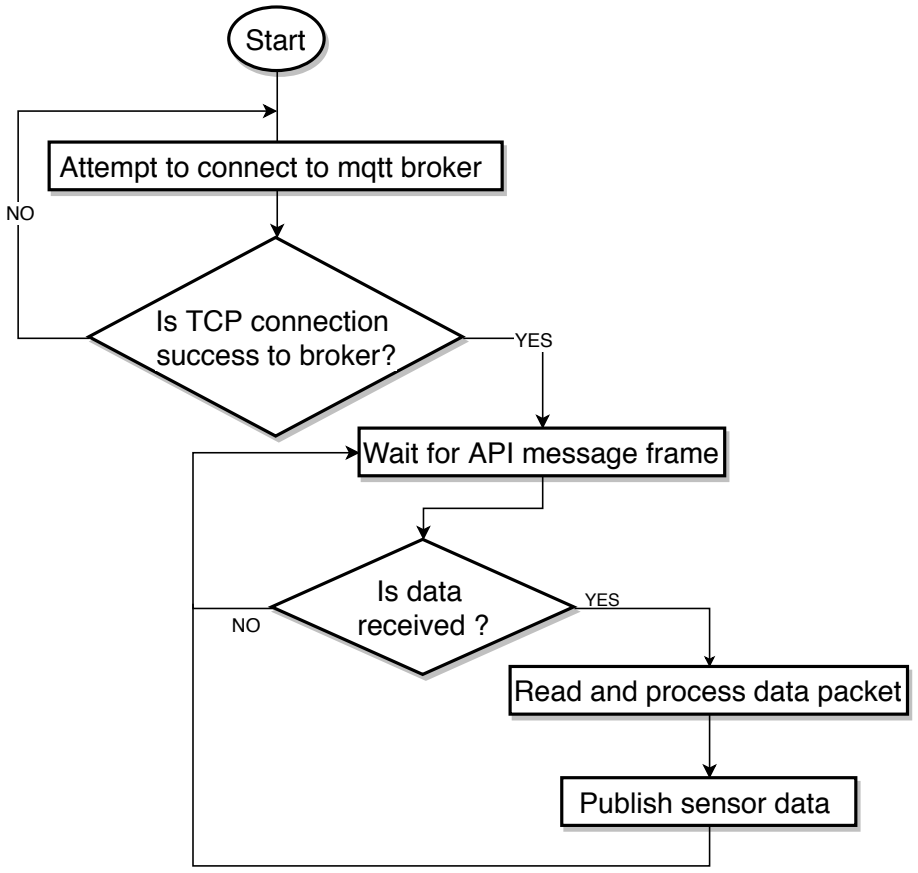

FIGURE 7.8: Flowchart of main loop of gateway node

7.3 Programming of Gateway Node

The basic process of the main loop of the gateway node is according to the flowchart shown in Figure 7.8.

The gateway node has the Raspberry Pi, and the XBee radio configured as full function device (i.e. coordinator). It does not have any sensors attached. After the gateway node has discovered its neighbors, it transmits the network setup messages to all the nodes in its neighbor table. The gateway node attempts to establish a TCP/IP connection with the MQTT broker. The connection is secured with user id and password provided from the "cloudmqtt" cloud. If connection is not successful, it tries again to establish the connection. If the connection is successful, main loop checks repetitively if there is any incoming sensor data received or not. It also subscribes to the incoming topic so as to enable the reception from the server for control operations (i.e. LED on or off). If there is any data received, it decodes the data using API format, and publish it to the server.

7.4 Proposed Data Aggregation Algorithm

End sensor nodes initially start sending sensor values (e.g. temperature and humidity). The intermediate node receives data, and calls `synchronize_and_perform_aggregation` routine, which basically synchronizes data reported from other sensor nodes with its own sensor values and checks for certain threshold value δ. If suppose the difference in sensor values is nearer to threshold value δ, it will satisfy the requirement of data aggregation. The threshold values (i.e. $\delta 1$; 5 % for temperature, and $\delta 2$; 5 % for humidity) are chosen. That means, if the absolute value of difference in last and current measurement of sensor values is less than or equal to this value, data will be aggregated (i.e. calculate the average value of measurements) and send to the next hop. If the difference is higher than this value and condition is not satisfied, the data is not processed with the aggregation subroutine, but transmitted immediately to the next hop.

We observed that with the same environmental conditions, within a fraction of seconds, the readings of temperature and humidity were fluctuating in the range of around 5 % with respect to previous readings. Therefore, the threshold value of 5 % has been chosen to calibrate the sensor outputs in order to get more precise values of both temperature and humidity. This also helps to reduce the frequent transmissions of packets. After all sensor nodes report data at least once, they will wait for the event triggered by the change in sensor data measurement with the locally aggregated data, or 1 hour periodic update, as described in Section 7.5.

Case I: Figure 7.9 shows the aggregated data (AD) after checking threshold value δ. Assume, sensor node 1 reads temperature 25 °C, sensor node 2 reads temperature 25 °C, sensor node 3 reads 26 °C, and sensor node 4 reads 25 °C, then the edges (1, 4), (2, 4) satisfy the threshold value of less than 5 % for temperature set by sensor node 4. The aggregated data at sensor node 4 (i.e. average data from nodes (1, 2, 4) or (1, 4) or (2, 4)) `AD_1` 25 °C. Similar calculation is done at sensor node 5.

Case II: Assume, sensor node 1 reads temperature 25 °C, Node2 reads temperature 29 °C, and sensor node 4 reads 26 °C then the edges (1, 4) satisfies the threshold value less than 5 %, but not the edge (2,4). The edge (2,4) has crossed more than 5 % threshold value set by sensor node 4. In this case, data is sent immediately without going through aggregation processing.

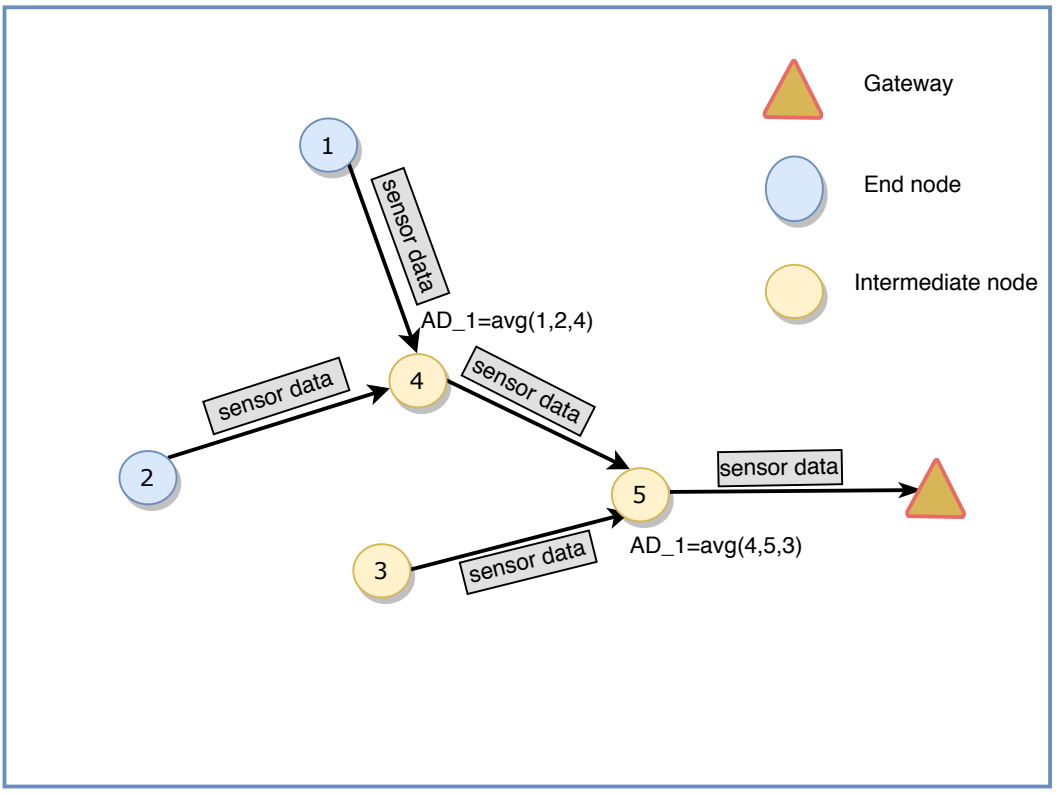

FIGURE 7.9: Data aggregation procedure

As per Figure 7.9, the total number of transmissions with the network without data aggregation is 11, and with data aggregation is 5. This approach is more useful when the number of sensor nodes increases significantly (i.e. more than 50) and frequency of data reporting is high at each node. We have also used threshold based event triggering data transmission to reduce the frequency of data reporting, as described in below Section 7.5.

7.5 Proposed Threshold Based Event Triggering Data Transmission

It was observed that there was almost the same measurement values continuously for a long time. Thus, if we start writing all measurements to the XBee radio, energy is wasted in transmitting the data that contains no variation, or a very little variation. Thus, sensor data can be locally aggregated before it exceeds the threshold set by the previous measurement.

In this method, the XBee end devices are first configured in pin hibernation mode from XCTU, as shown in Table 7.2, but not the router and the coordinator XBee devices. The reason is, when sleep is enabled in a router, the module joins the network as an end device, which disables its routing ability.

Parameters	Value
Sleep Mode (SM)	1
CTS (D7)	0

TABLE 7.2: Parameters of XBee for sleep mode

Arduino	XBee
Pin 9	Pin 9 (DTR/Sleep)
TX	RX
RX	TX
3.3 V	VCC (3.3 V)
GND	GND

TABLE 7.3: Arduino and XBee interfacing to put XBee in sleep mode

The end node micro-controller (i.e. ATMega328P) is programmed enabling sleep power save mode, but periodically wakes up the controller from the sleep power save mode after the expiry of watchdog timer [77] of 8 sec. After the sensor node successfully sends initial sensor reading, it checks if there is any change in the measurements between a previous and the current data. DTR/SLEEP pin is pulled up HIGH as shown in Table 7.3 (i.e. the XBee radio is put on sleep mode, and does not wake up unless the sensor node identifies the change in measurements). The threshold value δ is kept 5 %. The reason for putting this threshold value is described above in Section 7.4.

Initially, after the first measurement is successfully delivered to the next hop, we did not send the data at any other conditions unless there is a change detected in sensed data. This condition introduced some problems in our verification because it required to monitor the nodes manually to know they react to the change in the temperature or humidity, and they are still able to report the data. We verified the nodes periodically in hourly basis with the help of changing the environment (i.e. increased humidity by generating water vapor close to the nodes at least once in an hourly basis). We introduced a method to wake up the end sensor nodes at least once in 1 hour, and send

data to the server through a multi-hop. Using this method, we come to know that nodes are still active, and no failure has occurred.

Chapter 8

Experimental Results

In this section, we will provide the results achieved during testing and evaluation phases. In the first part, we will show the localization of sensor nodes where they are deployed. In the second part, we will present the output of web server data visualization and the data analytic results. In the third part, we will present the results of the tests conducted to verify re-routing functions when one of the intermediate nodes fails. In the last part, will provide the results of performance testing, such as packet reception ratio, network setup time, latency and battery lifetime tests.

8.1 Deployment Environment

FIGURE 8.1: A snapshot of how sensor nodes are deployed in a greenhouse (only few nodes are shown)

FIGURE 8.2: An outside view of deployment environment of greenhouse

FIGURE 8.3: An inside view of deployment environment of greenhouse

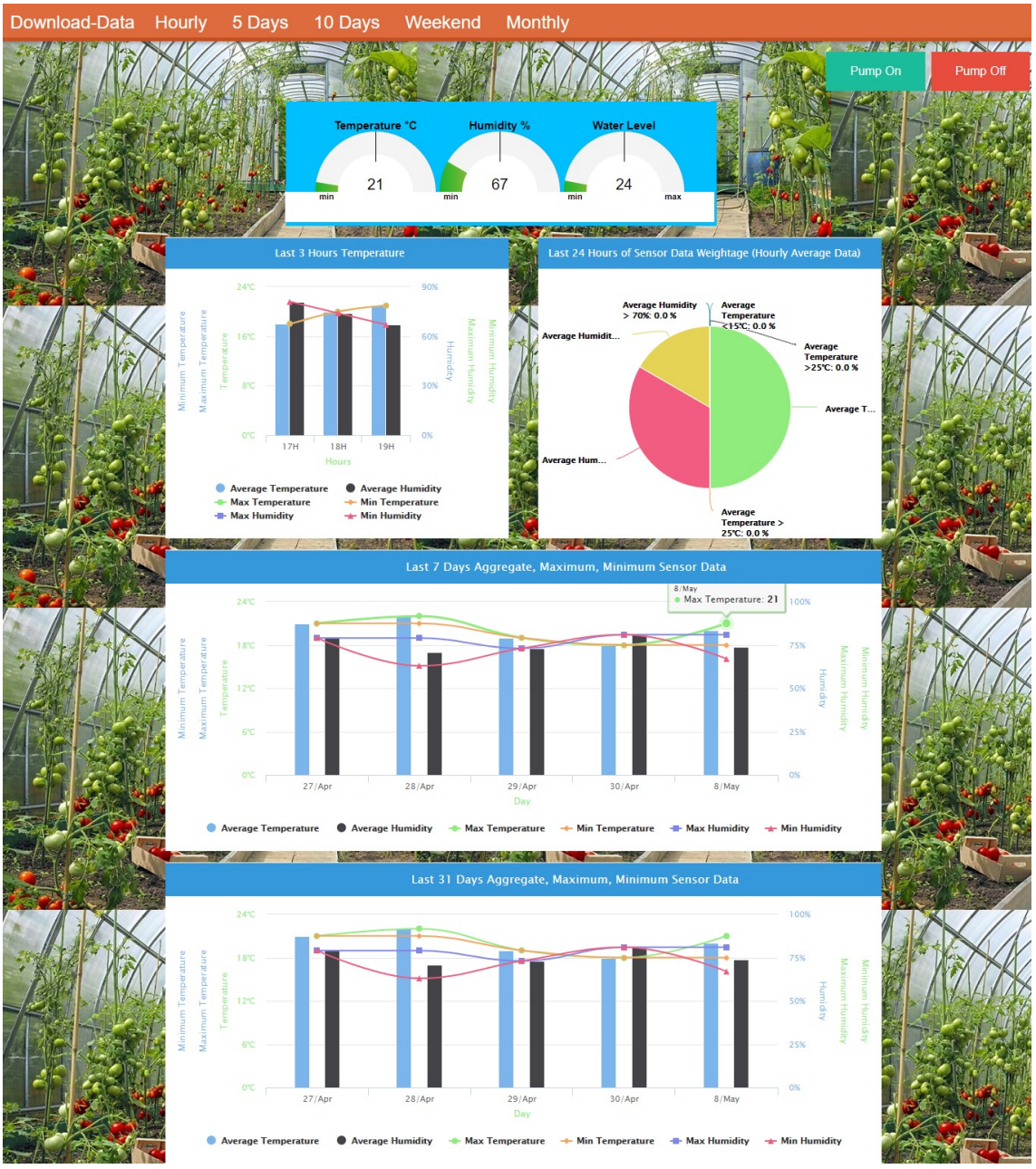

FIGURE 8.4: A snapshot of live web server data visualization and collection

The localization of test-bed setups are presented in Figure 8.1, which shows where the sensor nodes are deployed in the greenhouse. Figure 8.2 and Figure 8.3 show the outside and inside views of the greenhouse respectively. Each sensor device is connected with temperature and humidity sensors to get overall data within the greenhouse. Some sensor nodes are connected with moisture sensors and embedded inside the soil where plants have grown. They report data to the gateway node, which is connected to the internet via Wi-Fi. The gateway node is not connected to any sensors.

8.2 Output of Web Server Data Visualization and Analytic Results

We visualized data using various charting tools after processing it through descriptive data analytic services running in the back-end server. Using various charts, users can view the past trend of data. The data visualization is more detailed and informative for users to analyze it, as it gives the visual information in the forms of average, maximum, minimum values of sensor data on hourly, daily, weekly and monthly basis.

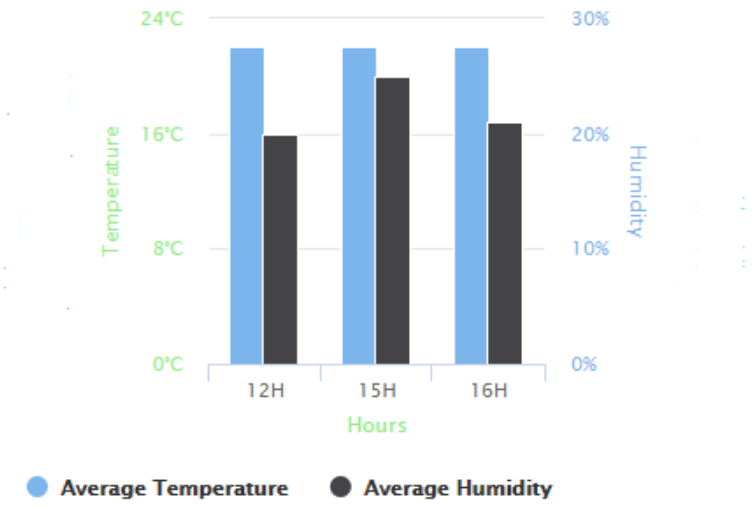

FIGURE 8.5: Hourly average temperature and humidity for the last 3 hours duration

The homepage of the web server where live sensor data is visualized is shown in Figure 8.4. We can view the real-time values of current temperature, humidity in the air and moisture level of the soil. The overall data can also be downloaded using .xls, .csv, .pdf, .png and tabular formats. A user can also view and download data on hourly, daily, weekly and monthly basis. Figure 8.5 shows that the average temperature for the last 3 hours (i.e. the average temperature was 22 °C, average humidity increased from 20 %RH to 25 %RH for 12 pm to 1 pm, but again decreased to 21 %RH for 2 pm). Figure 8.6 shows how a user can view the comparative variation of temperature and humidity for the last 24 hours (i.e. percentage of humidity or temperature below or above an alarming level, specified as in Chapter 1). Figure 8.7 shows the average temperature for the last 7 days was in the range of 22 °C to 23 °C, but the maximum temperature reached 23 °C, and the minimum temperature reaches 18 °C. The average humidity variation was from 45 %RH to 78 %RH, but maximum humidity reached 80 %RH.

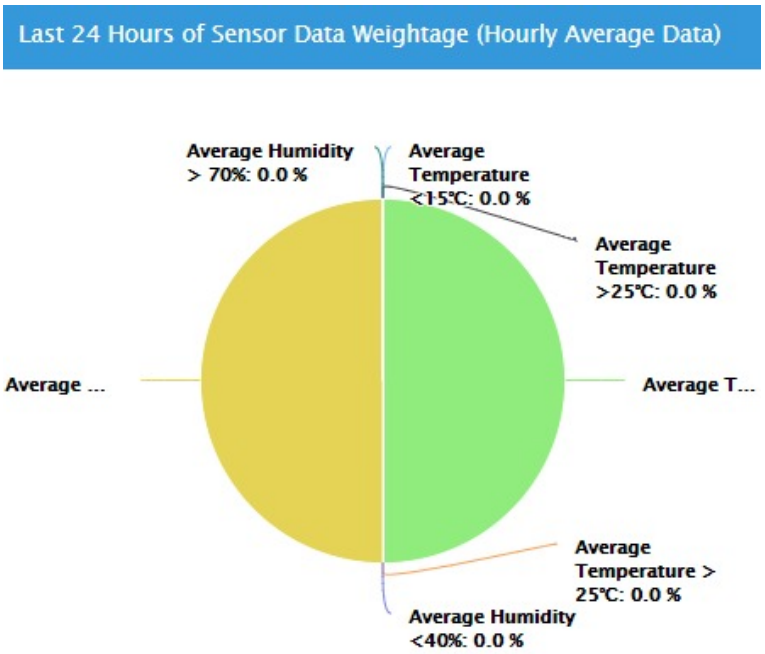

FIGURE 8.6: Last 24 hours of sensor data comparison

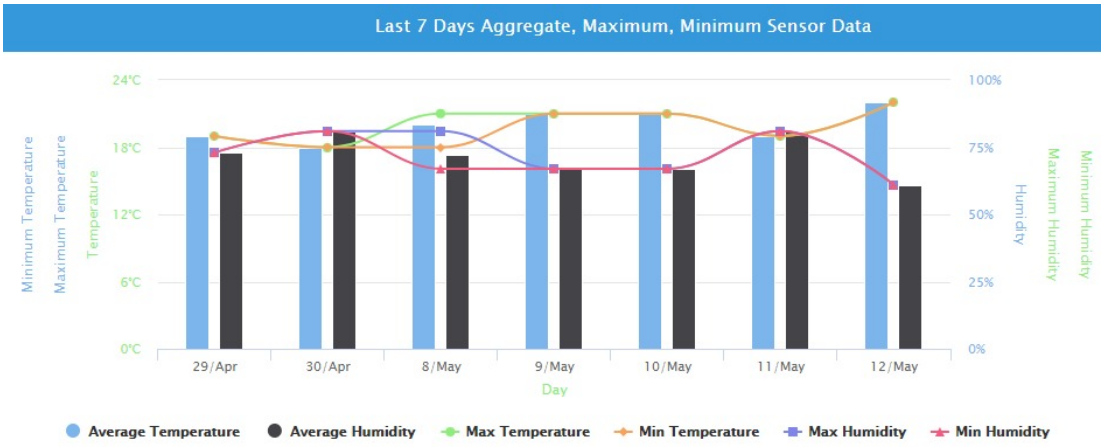

FIGURE 8.7: Daily average, maximum, minimum temperature and humidity for the last 7 days duration

Figure 8.8 shows the average daily temperature for the last 31 days was in the range of 22 °C to 24 °C, but maximum temperature reached 26 °C. The overall trend of temperature and humidity variation is shown in Figure 8.9, which can also viewed or downloaded using the start and end dates of user's choice. Figure 8.10 shows the average humidity distribution, and Figure 8.11 shows the average temperature distribution for the last week. The 3D-graphs are drawn for both humidity and temperature, which show the average hourly data for each day over a week. Figure 8.12 shows the output in Android widgets for temperature, humidity data.

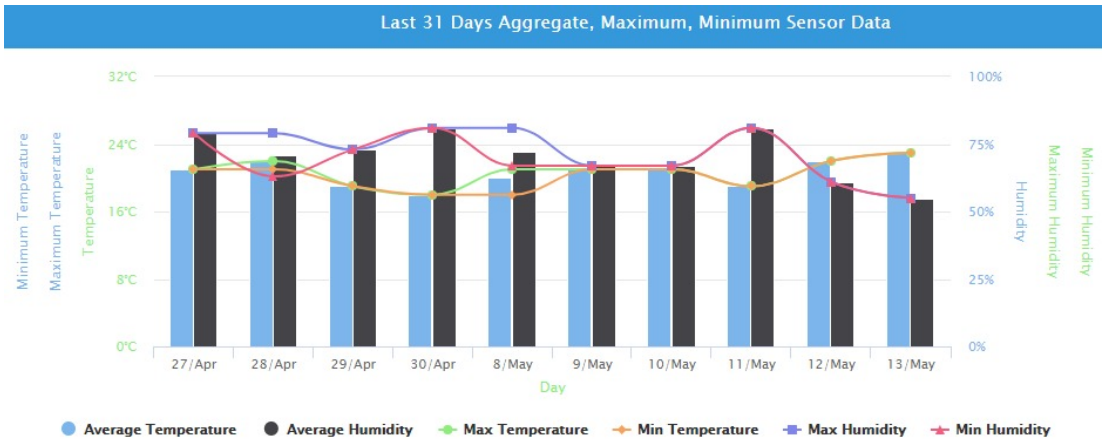

FIGURE 8.8: Daily average, maximum, minimum temperature and humidity for the last 31 days duration

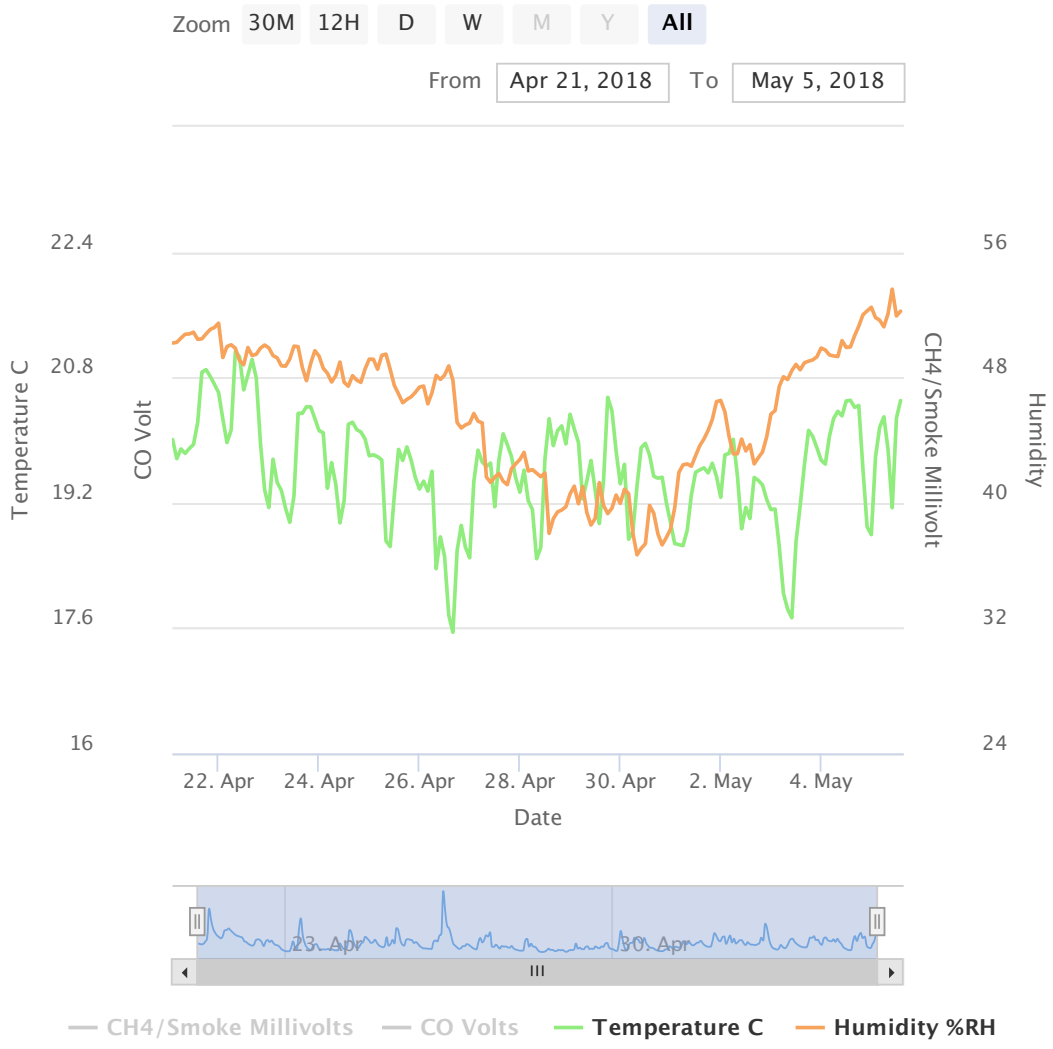

FIGURE 8.9: Overall trend of temperature and humidity data

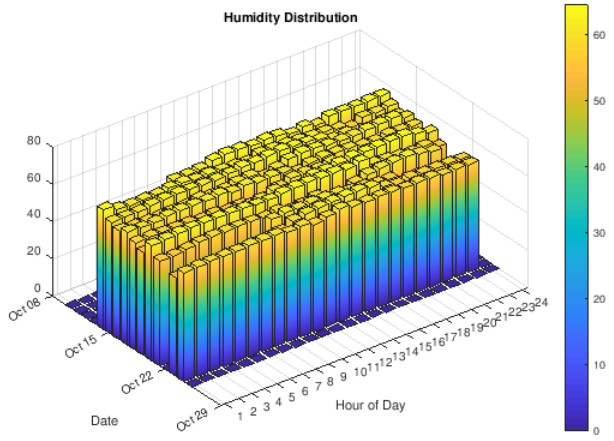

FIGURE 8.10: Humidity distribution during the last week

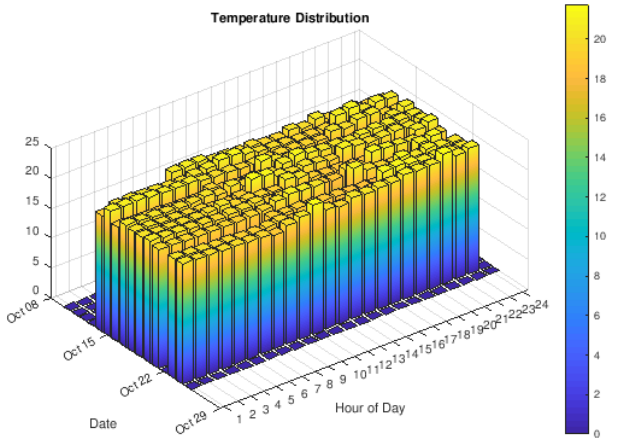

FIGURE 8.11: Temperature distribution during last week

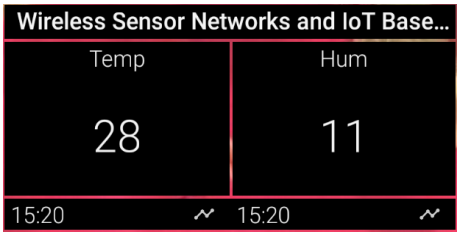

FIGURE 8.12: Temperature and humidity widgets in Android phones

8.3 Data Routing Test

This test has been done with 7 to 8 numbers of the XBees. To limit the communication range of the XBees, the line of sight is blocked, and the some XBees are used with no antennas.

8.3.1 Diagnostic Methods

- Two LED indicators (Red and Blue) and an inbuilt LED on the top of the XBee shield at each node are used for diagnostic purposes. All LED's, including inbuilt LED will flash once when a sensor node is powered on. Red LED will flash on for 2 seconds to indicate that nodes are discovered. Blue LED will flash on for 2 seconds to indicate that a sensor node has completed link quality estimation procedures with its neighbors. Blue LED will flash on 4 seconds to indicate that hops count is initialized.

- An inbuilt TX indicator LED in XBee Shield will flash on for one seconds to indicate that a sensor node has started sending the data.

- If the remote XBee did not receive our packet by sending ACK, an inbuilt LED will turn on for 2 seconds to indicate an error.

- To verify if the nodes are reacting to the change in humidity or temperature, we tested periodically with the help of generating water vapor, and putting it close to the nodes at different locations. We will get the changed data in the server if it is successful.

- For testing node failure and re-routing tests, a power source from the intermediate node is removed temporarily.

8.3.2 Sensor Node 3 or Sensor Node 4 Failure Case

Sensor nodes are kept in such a way that sensor nodes 1 and 2 can communicate only with sensor nodes 3 and 4, and are in the communication range, as shown in Figure 8.13. Once nodes get the number of hops count initialized, they start sending sensor data, as shown in Figure 8.13. Sensor nodes 3 and 4 both have the same number of hops count as 2. Sensor nodes 1 and 2 choose sensor node 4 as the next hop because of good link quality. Sensor node 4 receives data. Sensor node 4 starts looking for the next hop. Sensor node 4 does not send data back to sensor node 1, as it is the source address and does not count sensor node 2 and sensor node 3, as they have higher hops count than sensor node 5. Likewise, sensor node 3, after reading data, chooses sensor node 5 as the next hop.

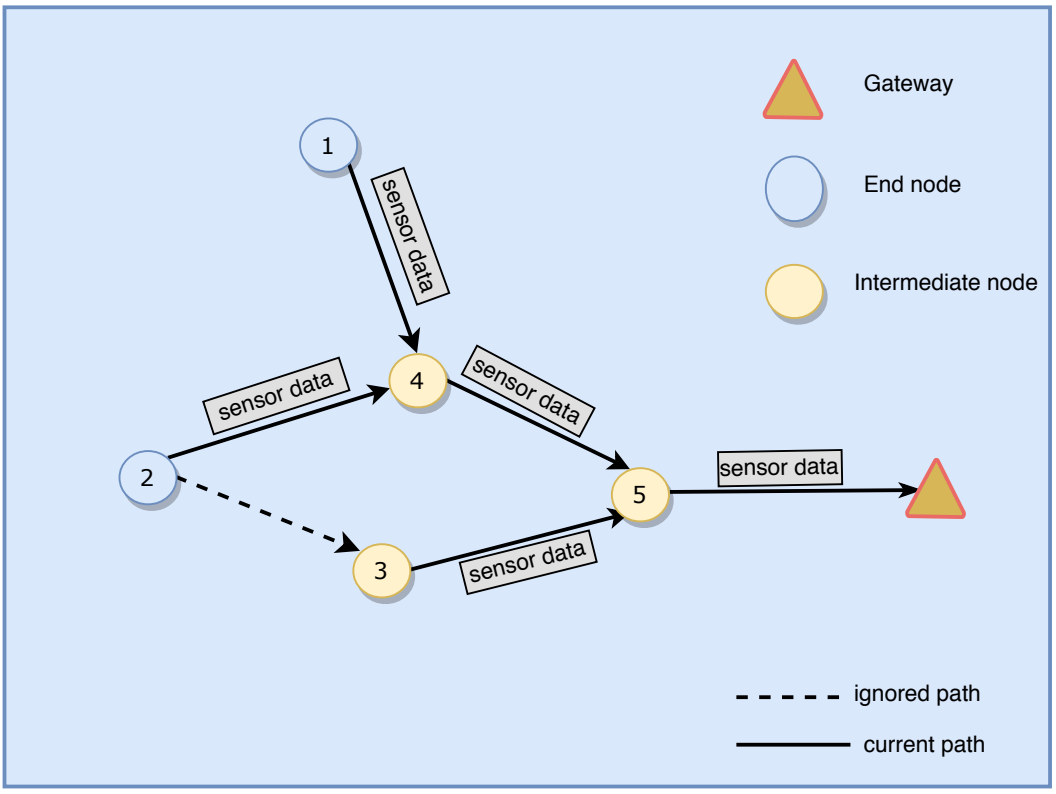

FIGURE 8.13: Initial routing

For testing node failure cases, sensor node 4 power source is removed, as shown in Figure 8.14. Sensor nodes 1 and 2 now start considering sensor node 3 as the next hop, after the first routing to Node4 fails and the retry count exceeds. The maximum number of retry counts is configured to 10. The number of hop count and the next hop for each node before and after Node4 failure are shown in Table 8.1.

Node	hops count	Next hop before node 4 failure	Next hop after node 4 failure
5	1	gateway	gateway
4	2	node 5	X
3	2	node 5	node 5
1	3	node 4	node 3
2	3	node 4	node 3

TABLE 8.1: Neighbor table before and after node 4 failure case

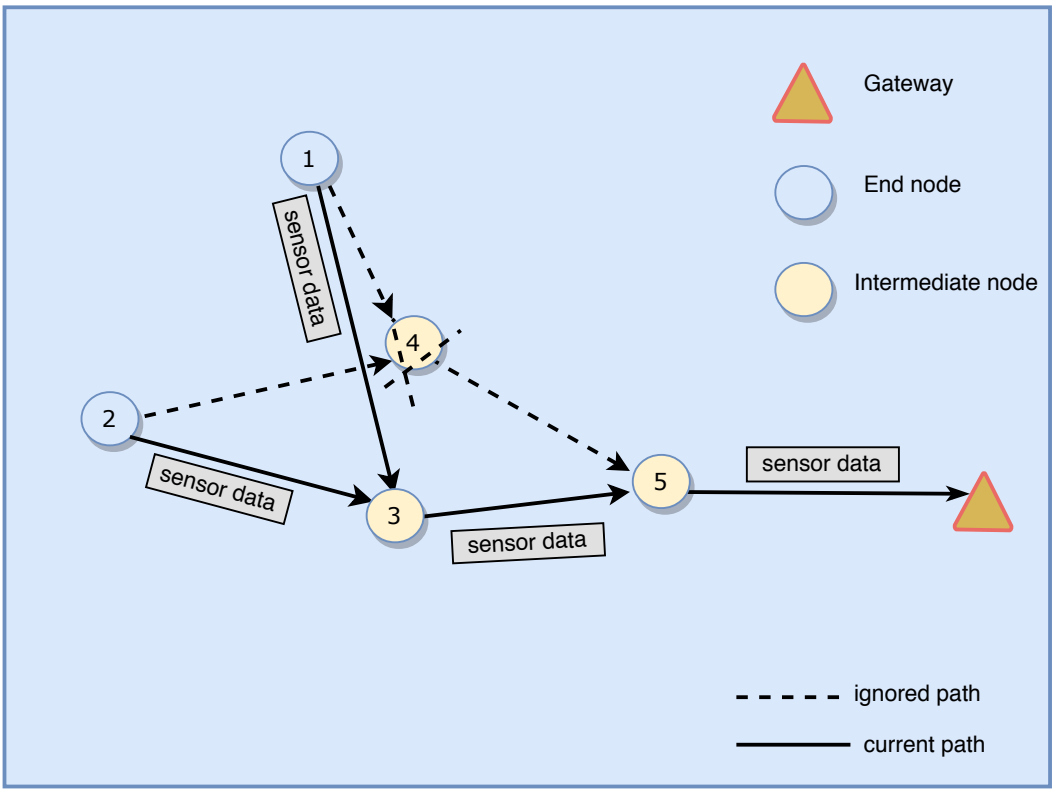

FIGURE 8.14: Node 4 failure case

8.3.3 Sensor Node 5 Failure Case

All child nodes do not find routes to the gateway according to the network of Figure 8.14 occurs and the entire network becomes non-functional, when sensor node 5 fails. None of nodes will be able to send data to the gateway nodes, as it is the only node which is coordinating to the gateway node. This situation can cause an entire network out of working. To avoid this situation, only option was to add one more node close to the gateway node. Thus, one more node is added close to the gateway node which can communicate to sensor nodes 3, 4 and 5. We added a new node (not shown in Figure) after configuring as the full function device, so that it allows other nodes to join the network, and can sense sensory data, and sends data to the gateway when no data forwarding is required. Sensor node 3 and 4 can consider that node (i.e. close to gateway) when sensor node 5 fails.

8.4 Scalability Test

8.4.1 A New Node Joins An Existing Network

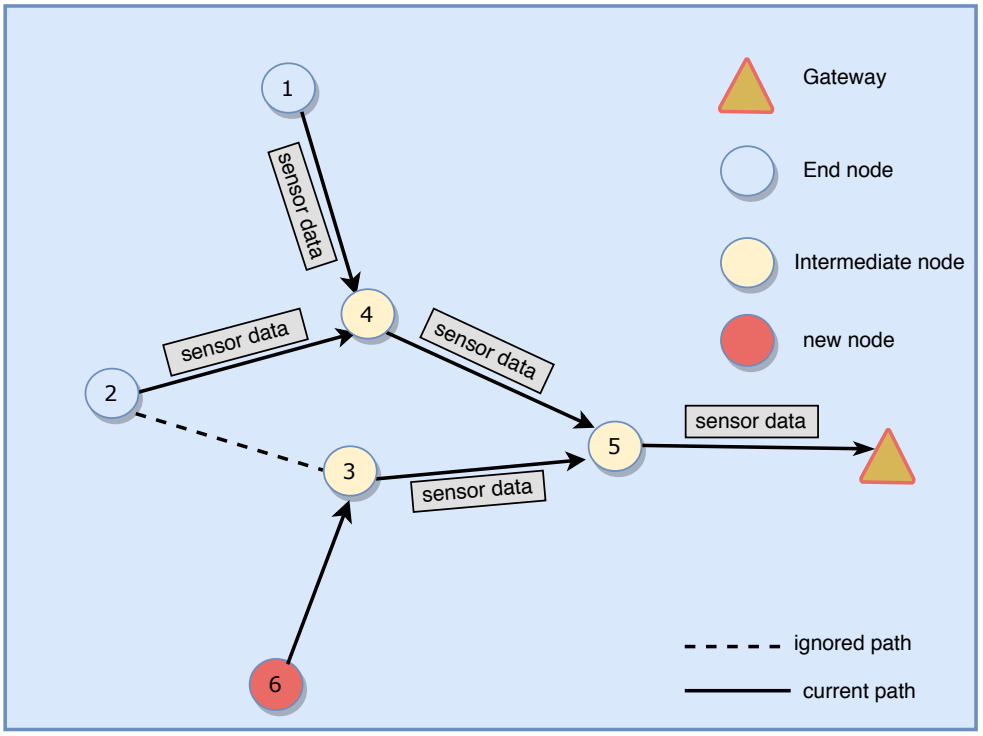

FIGURE 8.15: Node6 joins existing network

Whenever a new node tries to join the network, the network should automatically allow it to be integrated in the network, so that the entire network need not reset. This tests the scalability of the system. As shown in Figure 8.15, a new sensor node 6 is assigned the hop count of its neighbor node which is received from the response packet of LQE from node 3, plus one. Sensor node 3 allows sensor node6 to join the network, as sensor node 3 is configured as full function device. End nodes can not allow joining a new node for the expansion unless they are configured to full function device. For example, node2 does not allow a new node to join the network. In this case, we require one time changes to the existing node (i.e. configuring it as full-function device). This is unavoidable since we are configuring end devices (i.e. XBee as end device) to sleep during the time sensed data is not changed much with the previous measurement.

8.5 Performance Evaluation

8.5.1 Number of Successfully Acknowledged Packets

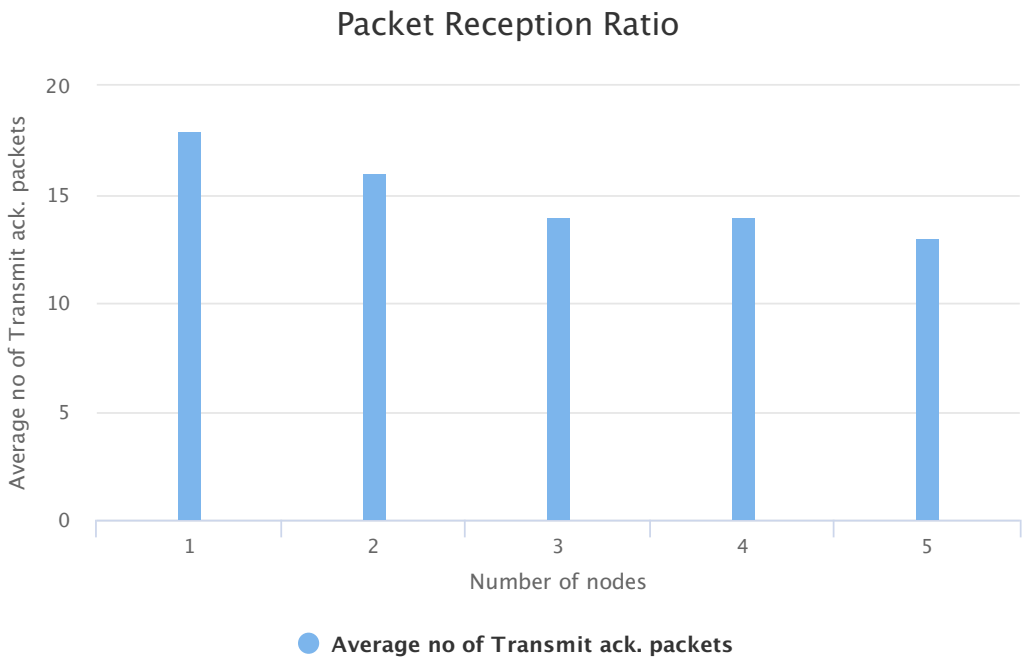

FIGURE 8.16: Number of ACK packets received during LQE procedure

Various factors, such as interference, radio link patterns, communication range, collisions affect the link quality. As a result, all the transmitted packets might not reach to the destination due to loss of the packets.

A test was conducted to observe the packet reception ratio (i.e. average number of received packets to the transmitted packets). Nodes were placed at different arbitrary distances at hops count of 1 from the gateway node. With an increase in the number of nodes, we observed the less average number of acknowledgement packets, as can be seen in Figure 8.16. We got the acceptable value of number of ACK packets (i.e. more than 5 ACK packets to consider a neighbor) even with the case of 5 neighboring sensor nodes. There is also a delay of 100ms in each transmission of link quality estimation packet. We observed that if we increase this delay more, more number of ACK packets were received successfully, but this will increase the network setup time. So, we kept this delay value to 100ms.

8.5.2 Network Setup Time

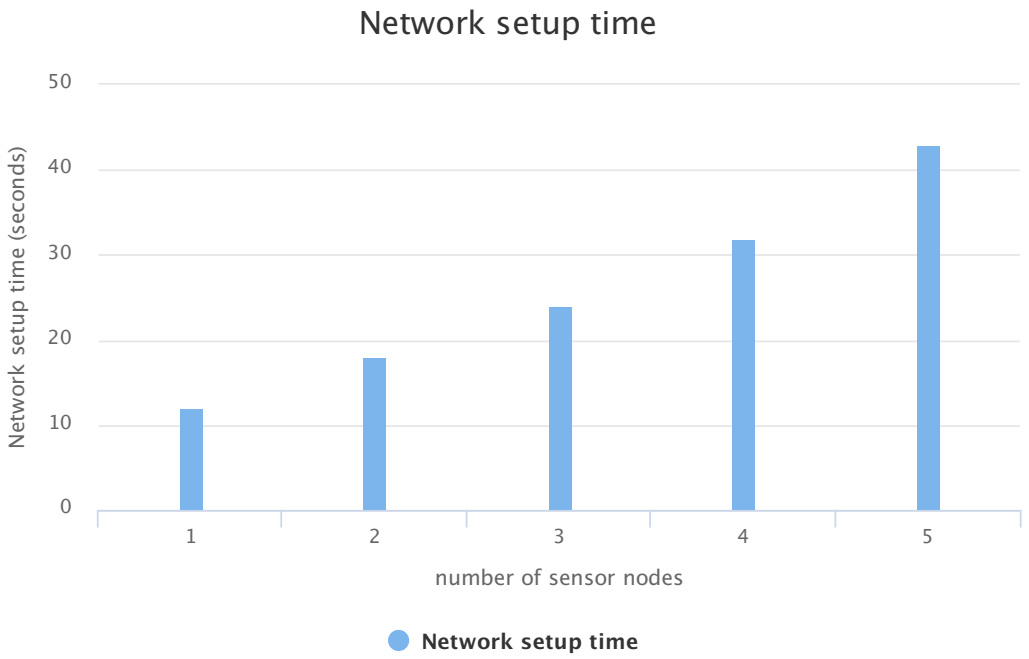

FIGURE 8.17: Network setup time

There is a network setup procedure introduced prior to starting the data packet routing. It is the time required to discover neighbor nodes, time to calculate LQE, and the time to reach network setup packet to the nodes, sent from the gateway node. There is also a wait time of 200 ms to read API frame packet. This introduces some overhead. So, a test was conducted to observe the time required while setting up the network (i.e. average time to propagate the required parameters through a network). When the number of nodes increases from 1 to 2, 3, 4, 5 or so on, it is expected that more time will be required to setup the network, but should not take too much time, which can degrade the performance.

Our observation shows that with an increase in the number of nodes, the average time to setup the network increases, as seen in Figure 8.17. The maximum time taken to setup the network in a 5 nodes multi-hop test was 43 seconds.

8.5.3 Latency

A multi-hop transmission introduces the latency, and it is also expected that there is some delay due to retry counts and packet processing at each node. We observed the

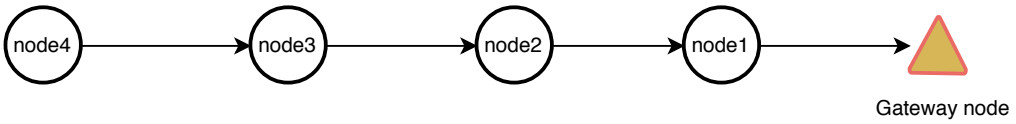

FIGURE 8.18: A multi-hop chain

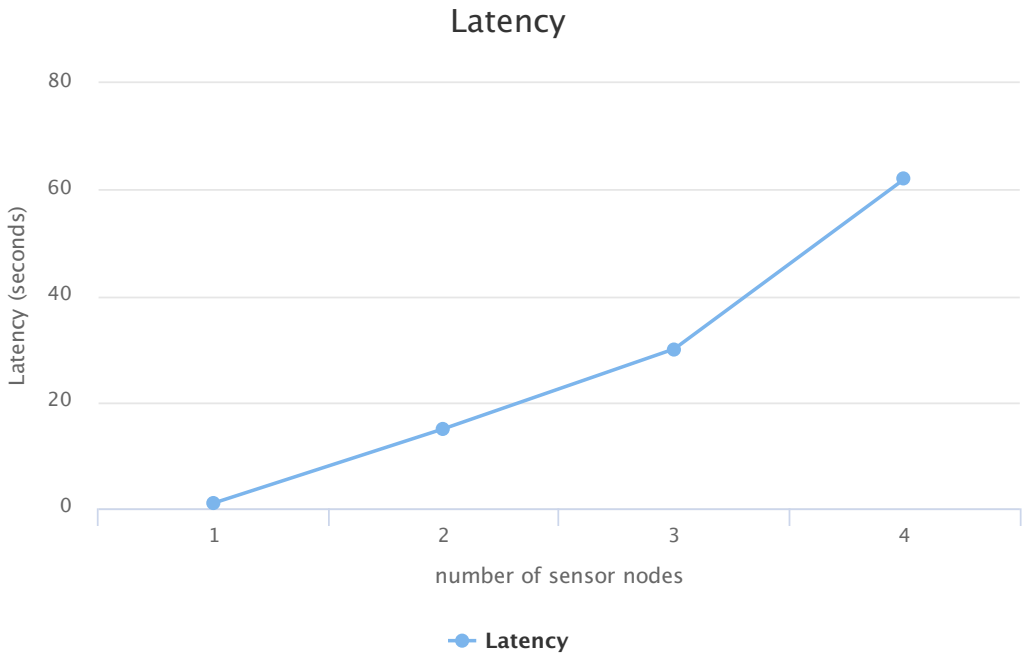

FIGURE 8.19: Latency

time taken by the nodes to transport the data packets to the gateway node using a multi-hop transmission. We conducted a 4 nodes multi-hop chain test by placing the nodes at arbitrary distances. A node which is far away from the gateway took longer time to transport the packets to the gateway node than compared to other nodes which are located near to the gateway node. The node which is just a one hop away from the gateway node took around 1 sec to send the data to it. The node which is 4 hops away from the gateway node took around 62 seconds to send the data to the gateway, or sometimes even more. This is because there is a added retry count functions (10 times with 200ms delay in between successive retries) at each node.

8.5.4 Number of Nodes Versus Number of Messages

If we expand our network (i.e. expand further more using a flat network according to Figure 8.15) by increasing more number of sensor nodes, it is expected that more number

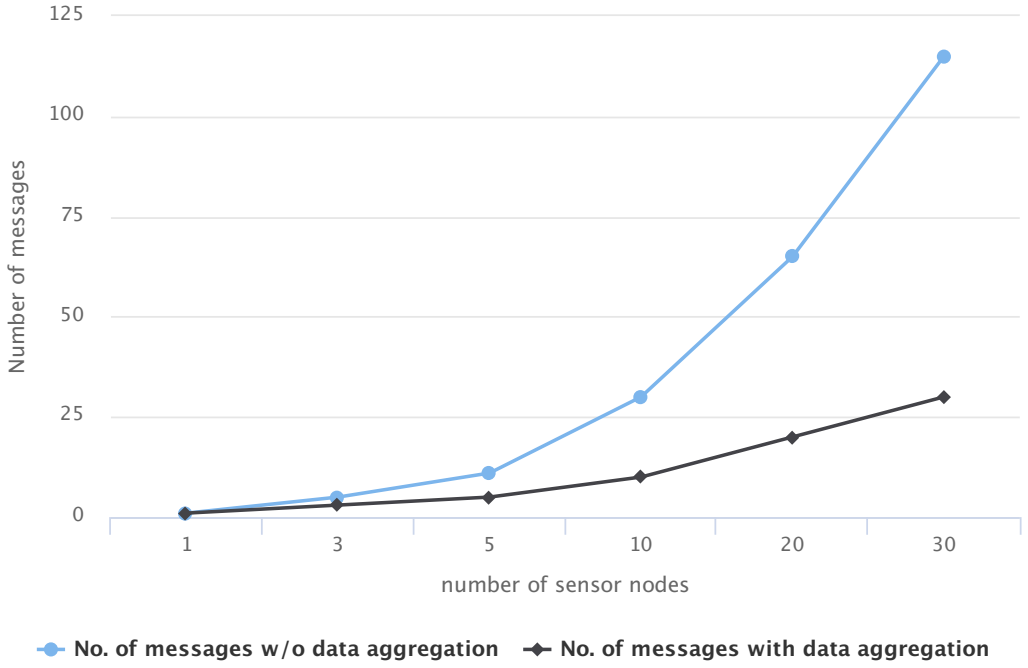

FIGURE 8.20: Number of nodes versus number of messages

of messages are generated during data routing. Figure 8.20 shows how an increase in sensor nodes affects the number of messages generated within the network just to report the data once, to the gateway node. As can be seen in Figure 8.20, without data aggregation approach, the number of messages increases with an increase in the number of nodes, and goes up sharply for a large number of sensor nodes because every node independently forwards the packets, reported from other nodes. This approach is more useful in saving energy when the network consists of a large number nodes and frequency of data reporting from each node is high. Also the number of messages generated will increase significantly if the time between consecutive samples transmitted is low (i.e. frequency of data reporting is high). Our event-based triggering method reduces the number of messages by lowering the time between consecutive samples transmission unless new sensed data is changed with locally aggregated data at each node.

Then, the energy consumption E per hop for sending 2 bytes data (i.e. 1 byte for temperature and 1 byte for humidity) plus 31 bytes of ZigBee API fixed header can be calculated as below.

$$E = E_{Tx} + E_{Rx} = K \times V \times I_{Tx} \times T_{Tx} + K \times V \times I_{Rx} \times T_{Rx} \tag{8.1}$$

where K is the total size of data in bytes, V is an operating voltage of the XBee module, I_{Tx} is a XBee transmitter current, T_{Tx} is a XBee transmitter time per bit, I_{Rx} is a XBee receiver current, T_{Rx} is a XBee receiver time per bit, E_{Tx} is energy consumption during transmission and E_{Rx} is energy consumption during reception, as shown in Table 8.2.

We can clearly see that by just reducing the number of messages from n to 1 at each node, we can save energy $(n-1)*E$, which is sufficient to send $(n-1)$ number of packets over the wireless link.

Parameters	Value
I_{Tx}	40 mA
I_{Rx}	40 mA
T_{Tx}	32 μs
T_{Rx}	32 μs
V	3.3 V

TABLE 8.2: Parameters of XBee Pro Series [78]

8.5.5 Battery Lifetime Prediction

The current consumed by the RF module can be computed as below [79].

$$I_{act(n)} = \frac{t_{onoff}I_{onoff} + t_{listen}I_{listen} + t_{Tx(n)}I_{Tx(n)}}{t_{active(n)}} \tag{8.2}$$

where t_{onoff} is the total time for activation and deactivation of the transceiver module, I_{onoff} is the current taken during this time, t_{listen} is listen time when transceiver is listening for a packet, I_{listen} is the current taken during this time, $I_{Tx(n)}$ is the current absorbed by the module during transmission of n bytes, and $t_{Tx(n)}$ is the time during this period.

$$t_{Tx(n)} = \frac{8 \times 31 + n}{r} \tag{8.3}$$

where r is the data rate (i.e. max data rate is 250 kbps), and ZigBee Mac layer adds a packet header overhead of 31 bytes with the payload, and $t_{active(n)}$ is the time for complete activity, which can be calculated as below.

$$t_{active(n)} = t_{onoff} + t_{listen} + t_{Tx(n)} \tag{8.4}$$

For end sensor nodes, the current at which the battery is drained considering T as the time between two consecutive transmissions and I_{sleep} as the sleep mode current, we can compute drain current by XBee as:

$$I_{RFdrain(n)} = \frac{t_{active(n)} \times I_{act(n)}}{T} + (1 - \frac{t_{active(n)}}{T}) \times I_{sleep} \tag{8.5}$$

But for router nodes, we have not enabled sleep mode in our design. So, $I_{RFdrain(n)}$ will be even higher because the XBee needs to listen for incoming packets for every t_{listen} time, drawing the current I_{listen}.

The lifetime (L) in hours of battery with capacity C (mAh) can be estimated as below.

$$L = \frac{C \times 1000}{I_{RFdrain(n)}} \tag{8.6}$$

As can be seen from above Equation 8.5, $I_{RFdrain(n)}$ will reduce if the time between two consecutive transmission (T) increases. As discussed in Chapter 7, our data transmission method uses event-triggering approach to write data to the XBee at least once in an hour. So, we have used T (3600 sec), unless data is changed frequently.

8.5.6 Battery Lifetime

Nodes	Predicted battery lifetime	Actual attery lifetime
For router nodes	$\sim$ 103 hours	$\sim$ 50 hours
For end sensor nodes	$\sim$ 155 hours	$\sim$ 120 hours

TABLE 8.3: Battery lifetime

A battery lifetime test was conducted using 11000 mAh Lithium Ion batteries. Table 8.3 shows the results for the router and end sensor nodes. The lifetime of end sensor nodes was around 120 hours and for router nodes, was around 50 hours. Arduino board which is not doing anything except being turned on will use about 50 mA of current from the power supply. If we use 11000 mAh battery, which will be around 220 hours of total battery lifetime at $I_{microcontroller}$ (i.e. current drawn by Arduino) of 50 mA per

hour. We also observed that even with the XBee or ATMega328P is put on SLEEP (i.e. end sensor nodes), there are other on-board peripheral, such as USB interface, voltage regulator, ICSP header, sensors which consume power, that is the reason the battery lifetime is shorter than the predicted one.

Chapter 9

Conclusion and Future Research Work

The IoT based greenhouse monitoring system designed in this thesis can be used as the greenhouse monitoring system. A wireless sensor network is designed which establishes a multi-hop path when a node starts sending sensory data to the gateway. The system has also incorporated cloud based web and mobile applications to enable online visualization of real-time streaming data. In addition, it is also equipped with the descriptive data analytic services that enable to view the data collected over a period of time, based on hourly, daily, weekly and monthly filters. The user can also retrieve the data of a specific period using the start and the end date. The user can set the alert level in the user interface of the developed application. The developed server application is able to produce the alarm when certain threshold values are exceeded.

A test was conducted to check the validity of rerouting algorithm when an occasional failure of intermediate nodes occurs, and it was passed. We also performed various performance evaluation results of the system related to network setup time, latency, battery consumption etc. The system is designed to minimize the number of transmissions during routing using data aggregation.

Sensors which we have used for this system can be used even for a large agriculture system outside greenhouse. We can also add more sensors easily to the existing system whenever required depending upon requirements with minimal changes, as the test-bed setup has interfaces to read data from many sensors. For a larger area deployment

in agriculture, more number of sensor nodes (i.e. more than 30, 50, 100 or so on, depending upon the requirement) are required. We can easily expand the network by adding many sensors nodes because we have used a flat network which makes it easily expandable. We can also secure data transported within WSN using encryption and decryption mechanisms because nodes can be vulnerable from outside attacks. But in the future, when the number of sensor nodes increases (i.e. more than 100), more than one gateway can be used to control the sensor network. An efficient algorithm about managing dynamic and static memory is required to maintain the routing table because the number of neighbors increases. The system can be integrated to the appliances to control humidity, supplying irrigation water and controlling temperature. Future research might focus on the implementation of the predictive data analytic services using Artificial Intelligence (AI) and Machine Learning (ML) to predict the future trend of air quality indices in the greenhouse. The predictive data analytic services are helpful in providing the future warning by detecting the anomaly in the measurements of air quality at early stage.

Bibliography

[1] S. Ivanov, K. Bhargava and W. Donnelly, "Precision Farming: Sensor Analytics," in IEEE Intelligent Systems, vol. 30, no. 4, pp. 76-80, 2015. doi:10.1109/MIS.2015.67

[2] Ministry of Agriculture, Food and Rural Affairs, Ontario, "Crop Protection Guide for Greenhouse Vegetable," Publication 835, 2016-2017.

[3] Ministry of Agriculture and Forestry, Alberta, "Management of the Greenhouse Environment". Available: http://www.agric.gov.ab.ca. [Last Accessed: December 25, 2017].

[4] Conover, C.A. and R.T. Poole., "Influence of light and fertilizer levels and fertilizer sources on foliage plants maintained under interior environments for one year," J. Amer. Soc. Hort. Sci. 106:571-574, 1991.

[5] Holler J, Tsiatsis V, Mulligan c, Karnouskos s,Avesand S. From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence. Amsterdam, The Netherlands: Elsevier, 2014.

[6] RFC 2616, Hypertext Transfer Protocol – HTTP/1.1 - IETF Tools. Available: https://tools.ietf.org/html/rfc2616. [Last Accessed: December 25, 2017].

[7] CoAP Technology. Available: http://coap.technology/. [Last Accessed: January 15, 2018].

[8] MQTT Version 3.1.1 Errata 01. Edited by Andrew Banks and Rahul Gupta. 10 December 2015. OASIS Approved Errata. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os.html. [Last Accessed: December 25, 2017].

[9] AES Security and Encryption. Available: http://docs.digi.com/display/RFKitsCommon/Security+and+encryption. [Last Accessed: December 25, 2017].

[10] Datagram Transport Layer Security Version 1.2. Available: https://tools.ietf.org/html/rfc6347. [Last Accessed: January 20, 2018].

[11] RESTful Web Services. Available: https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html. [Last Accessed: January 10, 2018].

[12] Standards for Wireless Communication. Available: https://nfc-forum.org/. [Last Accessed: December 25, 2017].

[13] Arduino Specification. Available: http://www.arduino.cc.[Last Accessed: February 01, 2017].

[14] Raspberry Pi 3 Model B. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b.[Last Accessed: December 25, 2017].

[15] DHT11 Specification. Available: https://www.adafruit.com/product/386. [Last Accessed: December 25, 2017].

[16] SHT10 Soil Temperature and Humidity Sensor. Available: https://www.adafruit.com/product/1298. [Last Accessed: January 20, 2017].

[17] Moisture Sensor. Available: https://www.sparkfun.com/products/13322. [Last Accessed: December 25, 2017].

[18] MQ-2 LPG Propane Methane Alcohol Sensor Hydrogen Smoke Gas Detection Module. Available: http://wiki.seeed.cc/Grove-Gas_Sensor-MQ2. [Last Accessed: January 20, 2017].

[19] MQ-7 Carbon Monoxide Sensor. Available: https://www.sparkfun.com/products/9403. [Last Accessed: December 25, 2017].

[20] Light Intensity Sensor. Available: http://wiki.seeed.cc/Grove-Light_Sensor. [Last Accessed: January 20, 2017].

[21] Raspberry Pi 3 Specification. Available: https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1_0.pdf. [Last Accessed: December 25, 2017].

[22] XBee-PRO XBee Modules (S2). Available: www.digi.com. [Last Accessed: December 25, 2017].

[23] Noobs Installation Guide, Last Accessed, December 2017. Available: https://www.raspberrypi.org/downloads/noobs/. [Last Accessed: December 25, 2017].

[24] Arduino XBee Shield, Last Accessed, December 2017. Available: http://wiki.seeed.cc/XBee_Shield_V2.0. [Last Accessed: January 16, 2018].

[25] XBee-PRO Explorer USB Dongle. Available: https://www.sparkfun.com/products/11697. [Last Accessed: December 25, 2017].

[26] Arduino IDE. Available: https://www.arduino.cc/en/Main/Software. [Last Accessed: February 01, 2017].

[27] DHT11 Library for Arduino. Available: https://github.com/adafruit/DHT-sensor-library. [Last Accessed: January 10, 2018].

[28] SHT10 Sensor Library for Arduino. Available: https://github.com/practicalarduino/SHT1x. [Last Accessed: December 25, 2017].

[29] Raspberry Pi OS Installation Procedure. Available: http://learn.adafruit.com/setting-up-a-raspberry-pi-with-noobs. [Last Accessed: January 10, 2018].

[30] Raspberry Pi 3 Model B. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Last Accessed: February 01, 2018].

[31] XCTU Software, Last Accessed, December 2017. Available: https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu. [Last Accessed: January 14, 2018].

[32] Node.js. Available: https://nodejs.org. [Last Accessed: December 25, 2017].

[33] MongoDB. Available: https://www.mongodb.com. [Last Accessed: January 14, 2018].

[34] Express Web Application Framework. Available: https://expressjs.com. [Last Accessed: January 14, 2018].

[35] HyperText Markup Language. Available: https://html.com. [Last Accessed: January 14, 2018].

[36] jQuery. Available: http://jquery.com./ [Last Accessed: December 25, 2017].

[37] AngularJS Framework. Available: https://angularjs.org. [Last Accessed: December 25, 2017].

[38] Socket.IO. Available: https://socket.io/. [Last Accessed: January 11, 2018].

[39] Thingspeak Cloud. Available: https://thingspeak.com. [Last Accessed: January 11, 2018].

[40] Android Studio. Available: https://developer.android.com/studio/index.html. [Last Accessed: January 11, 2018].

[41] ZigBee Alliance. Available: http://www.zigbee.org. [Last Accessed: January 11, 2018].

[42] Drew Gislason, Zigbee Wireless Networking, Newnes, Newton, MA, 2008.

[43] ZigBee Generic API Frame. Available: http://docs.digi.com/display/RFKitsCommon/XBee+API+mode. [Last Accessed: January 11, 2018].

[44] Alvarez-Campana, M.; López, G.; Vázquez, E.; Villagrá, V.A.; Berrocal, J. Smart CEI Moncloa: An IoT-based Platform for People Flow and Environmental Monitoring on a Smart University Campus. Sensors 2017, 17, 2856.

[45] Harsharn S. Grewal, Basant Maheshwari, Sophie E. Parks, Water and nutrient use efficiency of a low-cost hydroponic greenhouse for a cucumber crop: An Australian case study, Agricultural Water Management, Volume 98, Issue 5, 2011, Pages 841-846, ISSN 0378-3774.

[46] Yiming Zhou, Xianglong Yang, Liren Wang, and Yibin Ying, "Wireless Design of Low-Cost Irrigation System Using ZigBee Technology." in the proceedings of the International Conference on Networks Security, Wireless Communications and Trusted Computing, 2009. NSWCTC '09. (Volume:1), Wuhan, Hubei, DOI: 10.1109/NSWCTC.2009.231, 25-26 April 2009, pp. 572 - 575.

[47] A. Bletsas, A. Vlachaki, E. Kampianakis, G. Sklivanitis, J. Kimionis, K. Tountas, M. Asteris, and P. Markopoulos, "Towards precision agriculture: Building a soil wetness multi-hop wsn from first principles," in Second International Workshop in Sensing Technologies in Architecture, Forestry and Environment (ECOSENSE), Belgrade, Serbia, Apr. 2011.

[48] Prihtiadi, Hafizh; Djamal, Mitra. The Reliability of Wireless Sensor Network on Pipeline Monitoring System. Journal of Mathematical and Fundamental Sciences, [S.l.], v. 49, n. 1, p. 51-56, apr. 2017. ISSN 2338-5510.

[49] G. Werner-Allen, P. Swieskowski and M. Welsh, "MoteLab: a wireless sensor network testbed," IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005., Boise, ID, USA, 2005, pp. 483-488.

[50] J. Yang, C. Zhang, X. Li, Y. Huang, S. Fu, M.F. Acevedo. Integration of wireless sensor networks in environmental monitoring cyber infrastructure.Wireless Networks, Springer/ACM, Volume 16, Issue 4, pp. 1091-1108, May 2010.

[51] B. F. Spencer, Jr., and C.-B. Yun, "Wireless sensor advances and applications for civil infrastructure monitoring," Newmark Structural Engineering Laboratory, Univ. Illinois at Urbana-Champaign, Urbana, IL, USA, Tech. Rep. NSEL-024, 2010.

[52] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: An acquisitional query processing system for sensor networks," ACM Trans. Database Syst., vol. 30, no. 1, pp. 122–173, Mar. 2005.

[53] R. Mueller, G. Alonso, and D. Kossmann, "SwissQM: Next generation data processing in sensor networks," in Proc. Innov. Data Syst. Res., vol. 7, 2007, pp. 1–9.

[54] S. G. Nikhade, "Wireless sensor network system using Raspberry Pi and zigbee for environmental monitoring applications," Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM), 2015 International Conference on, Chennai, 2015, pp. 376-381.

[55] Contiki: The Open Source OS for the Internet of Things. Available: http://www.contiki-os.org/. [Last Accessed: January 11, 2018].

[56] Sharmila, R.; Thanuja, R.; Umamakeswari, A.. Data Aware Aggregation Technique for Environmental Monitoring in Wireless Sensor Networks. Indian Journal of Science and Technology, [S.l.], dec. 2016. ISSN 0974 -5645.

[57] F. Marcelloni and M. Vecchio, "An efficient lossless compression algorithm for tiny nodes of monitoring wireless sensor networks," Comput. J., vol. 52, no. 8, pp. 969–987, 2009.

[58] Y. Liang and W. Peng, "Minimizing energy consumptions in wireless sensor networks via two-modal transmission," SIGCOMM Comput. Commun. Rev., vol. 40, no. 1, pp. 12–18, 2010.

[59] T. Srisooksai, K. Kaemarungsi, P. Lamsrichan and K. Araki, "Energy Efficient Data Compression in Clustered Wireless Sensor Networks using Adaptive Arithmetic Coding with Low Updating Cost," International Journal of Information and Electronics Engineering, vol. 1, no. 1 , pp. 85-93, July 2011.

[60] S. Hedetniemi and A. Liestman, "A Survey on Gossiping and Broadcasting in Communication Networks", IEEE Network, vol. 18, no. 4, pp. 319 349, 1998.

[61] W. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks", Proceedings of the 5th International Conference on Mobile Computing and Networking (Mobicom), 15 - 19 August 1999, Seattle, USA, pp. 174 - 185, August 1999.

[62] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks", Proceedings of the 6th International Conference on Mobile Computing and Networking (Mobicom), Boston, USA, pp. 56 - 67, August 2000.

[63] D. Braginsky and D. Estrin, "Rumor Routing Algorithm for Sensor Networks", Proceedings of the first Workshop on Sensor Networks and Applications, 28 September 2002, Atlanta, USA, pp. 22 - 31, 2002.

[64] F. Ye, A. Chen, S. Lu and L. Zhang, "A Scalable Solution to Minimum Cost Forwarding in Large Sensor Networks", Proceedings of the IEEE International Conference on Computer Communication and Networks (ICCCN), 15 - 17 October 2001, Phoenix, USA, pp. 304 - 309, 2001.

[65] Xia L., Chen X. and Guan X, "A New Gradient-Based Routing Protocol in Wireless Sensor Networks", vol 3605. Springer, Berlin, Heidelberg, ICESS 2004.

[66] R. C. Shah and J. Rabaey, "Energy Aware Routing for Low Energy Ad Hoc Sensor Networks", Proceedings of the IEEE Wireless Communications and Networking Conference, 17 - 21 March 2002, Orlando, USA, vol. 1, pp. 350 355, 2002.

[67] W. Heinzelman, A. Chandrakasan and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks", Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS), 31 January – 4 February 2005, Hawaii, USA, pp. 1 - 10, 2000.

[68] M. Hempel, H. Sharif and P. Raviraj, "HEAR-SN: A New Hierarchical Energy-Aware Routing Protocol for Sensor Networks", Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS), 3 - 6 January 2005, Hawaii, USA, pp. 324a, 2005.

[69] A. Manjeshwar and D. Agrawal, "TEEN: A Protocol for Enhanced Efficiency in Wireless Sensor Networks", Proceedings of the 1st International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing, San Francisco, California, USA, April 2001.

[70] Baccour, N., Koubaa, A., Ben Janaa, M., Youssef, H., Zuinga, M., Alves, M., "A Comparative Simulation Study of Link Quality Estimators in Wireless Sensor Networks", 17th IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, London, UK, pp. 1 10, 2009.

[71] Battery Discharge Graph. Available: http://batteryuniversity.com/learn/article/lithium_based_batteries. [Last Accessed: January 10, 2018].

[72] Battery Voltage Sensor. Available: http://www.dx.com/. [Last Accessed: January 10, 2018].

[73] Paho MQTT Client. Available: https://pypi.python.org/pypi/paho-mqtt/1.1. [Last Accessed: January 14, 2018].

[74] MQTT Version 3.1.1 Errata 01. Edited by Andrew Banks and Rahul Gupta. 10 December 2015. OASIS Approved Errata. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os.html. [Last Accessed: December 25, 2017].

[75] XBee Library for Arduino. Available: https://github.com/andrewrapp/xbee-arduino. [Last Accessed: January 10, 2018].

[76] Python Library for XBee. Available: https://github.com/digidotcom/python-xbee/blob/master/examples/network/DiscoverDevicesSample/DiscoverDevicesSample.py. [Last Accessed: January 10, 2018].

[77] Datasheet ATMega328P. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf. [Last Accessed: January 18, 2018].

[78] ZigBee Alliance. Available: http://www.zigbee.org. [Last Accessed: January 18, 2018].

[79] Piyare, Rajeev, Lee, Seong-Ro. (2013). Performance Analysis of XBee ZB Module Based Wireless Sensor Networks. International Journal of Scientific and Engineering Research. 4.

[80] Highchart. Available: https://www.highcharts.com/. [Last Accessed: January 25, 2018].

[81] Build an Android App Widget. Available: https://developer.android.com/guide/topics/appwidgets/index.html. [Last Accessed: January 18, 2018].

[82] ZigBee Generic API Frame. Available: http://docs.digi.com/display/RFKitsCommon/XBee+API+mode. [Last Accessed: January 18, 2018].