# DEVELOPMENT AND TESTING OF A MESH ROUTING PROTOCOL FOR IERMON (INDIAN ENVIRONMENTAL RADIATION MONITORING NETWORK)

*A Project Report*

*submitted by*

**G. PRIYANKA REDDY**

*in partial fulfilment of the requirements*
*for the award of the degree of*

**MASTER OF TECHNOLOGY**


**DEPARTMENT OF ELECTRICAL ENGINEERING**
**INDIAN INSTITUTE OF TECHNOLOGY MADRAS.**

**MAY 2015**

# THESIS CERTIFICATE

This is to certify that the thesis titled **DEVELOPMENT AND TESTING OF A MESH ROUTING PROTOCOL FOR IERMON (INDIAN ENVIRONMENTAL RADIATION MONITORING NETWORK)** , submitted by **G. PRIYANKA REDDY**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by her under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

**Dr. Boby George**
Project Guide
Associate Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

**Prof.Venkatesh Ramaiyan**
Project Co-Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

# ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my project guide Dr. Boby George for his guidance, invaluably constructive criticism and constant support through the course of this project. I am thankful to my co-guide Dr. Venkatesh Ramaiyan and Bhabha Atomic Research Centre (BARC) guide Dr. A. Vinod Kumar for their valuable advices and motivation.

I also want to thank Bhabha Atomic Research Centre (BARC) for their financial support granted for the project.

My heartfelt thanks to all my fellow research scholars in Measurements and Instrumentation lab, especially Srinivas and Anish for being a helping hand throughout the project. I appreciate Anusha, Arathy, Biswarup, Prashanth and Sreenath for their support during my work. I also wish to thank my room-mates for their encouragement. I would also extend my thanks to IERMON group, BARC for their support.

Finally, a special thanks to my parents and family for their support and I am grateful to the God for the good health and well-being that were necessary to complete this project.

# ABSTRACT

KEYWORDS: IERMON ; Densification of ERM; MetroNet; IEEE 802.15.4; Routing Protocols, Sensor data, central aggregator; Preset threshold.

The existing IERMON (Indian Environmental Radiation Monitoring Network) system has many ERMs (Environmental Radiation Monitors), deployed countrywide. Each ERM is having radiation sensors and ability to communicate the measured dose level to IERMON central station. ERM is operated by battery with solar power recharge backup and has GSM based data communication. GSM communication consumes large amount of power. This restricts the size beyond which the battery and solar panel size can be reduced further to minimize the ERM size and cost.

For densification of ERM in vulnerable locations or in metropolitan cities (under IERMON MetroNet program), it will be very useful if miniaturized-low power and low cost modules can be employed using wireless communication. The MetroNet with wireless communication can then be linked to one of the ERM in the IERMON having GSM communication to act as RHS (Regional-Hub-Station). This will reduce the installation, maintenance and operational cost.

The proposed system for MetroNet employs IEEE 802.15.4 standard as the base communication protocol. Three different routing protocols and their pros and cons have been studied in this work. Among the three tested algorithms, one of the routing algorithm is developed in-house. Before deploying and testing the protocols, range and delay aspects of the components with the space of deployment were assessed. The implemented protocols have been enabled to transmit the sensor data to a central aggregator in the MetroNet called as coordinator, only if a preset threshold is crossed. Apart from transmitting sensor data at regular intervals on crossing threshold, the routing algorithm which has been developed as part of this work also detects the path taken by the data to reach the data collector and ensures that the node with low battery levels will not be overused.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABBREVIATIONS

| | |
|---|---|
| **IERMON** | Indian Environmental Radiation Monitoring Network |
| **ERM** | Environmental Radiation Monitors |
| **RHS** | Regional-Hub-Station |
| **GSM** | Global System for Mobile Communications |
| **IEEE** | Institute of Electrical and Electronics Engineering |
| **AODV** | Ad hoc On-Demand Distance Vector Routing |
| **WPAN** | Wireless Personal Area Network |
| **PHY** | Physical |
| **MAC** | Medium Access Control |
| **BPSK** | Bipolar Phase Shift Keying |
| **O-QPSK** | Orthogonal Quadrature Phase Shift Keying |
| **ASK** | Amplitude Shift Keying |
| **ED** | Energy Detection |
| **CS** | Carrier Sense |
| **CCA** | Clear Channel Assessment |
| **RSS** | Received Signal Strength |
| **LQI** | Link Quality Indicator |
| **GTS** | Guaranteed Time Slots |
| **CSMA-CA** | Carrier Sense Multiple Access with Collision Avoidance |
| **RREQ** | Route Request |
| **RREP** | Route Reply |
| **RRER** | Route Error |
| **AES** | Advanced Encryption Standard |
| **PAN ID** | Personal Area Network Identification |
| **NL** | Neighbor List |

# NOTATION

| | |
|---|---|
| C0 | Cordinator |
| R# | Router Number |
| S1 | Series 1 |
| S2 | Series 2 |
| $T_r$ | time-stamp at which XBee receives back the transmitted packet |
| $T_t$ | time-stamp at which XBee transmits the packet |

# CHAPTER 1

# INTRODUCTION

Radiation is not harmful if under limit. The natural radiation has always been a part of human life and is of either terrestrial origin or cosmic origin. The terrestrial radiation is due to the presence of naturally occurring radioactive substances in earth's crust. Cosmic radiation comes through the earth's atmosphere, from the sun and galaxies.

IERMON was started with an agenda of monitoring environmental radiation throughout the country, for an early detection and activation of nuclear emergency measures. It also aims at eliminating the false propaganda of alarmingly increasing radiation levels due to the establishment of nuclear reactors. IERMON system has many ERMs, deployed countrywide. Each ERM is having radiation sensors and ability to communicate the measured dose level to IERMON central station.

Generally, Wireless Sensor Networks(WSNs) are used to monitor environmental conditions. A WSN is a system that consists of thousands of very small stations called sensor nodes. The nodes are meant to sense, compute and communicate, relying on a battery to stay active. And for these battery powered nodes to operate unattended for long time, they should consume low power which implies short range.

The existing IERMON has GSM based data communication and hence consumes large amount of power. Even though it uses a solar panel for recharging its battery, large power consumption restricts the reduction of size of the solar panel employed[1].

Hence, there is a rising need for a miniaturized, low power short range communication protocol based WSN setup as a part of MetroNet, which will act as an extension of the IERMON, reporting the sensor data to the RHS.

Sensor data in IERMON is the radiation dose level which is in text format, hence a low data rate communication protocol will be sufficient. Zeroing all the requirements, a communication protocol with less cost, low power consumption, short range, small size and low data rate has to be chosen for building MetroNet.

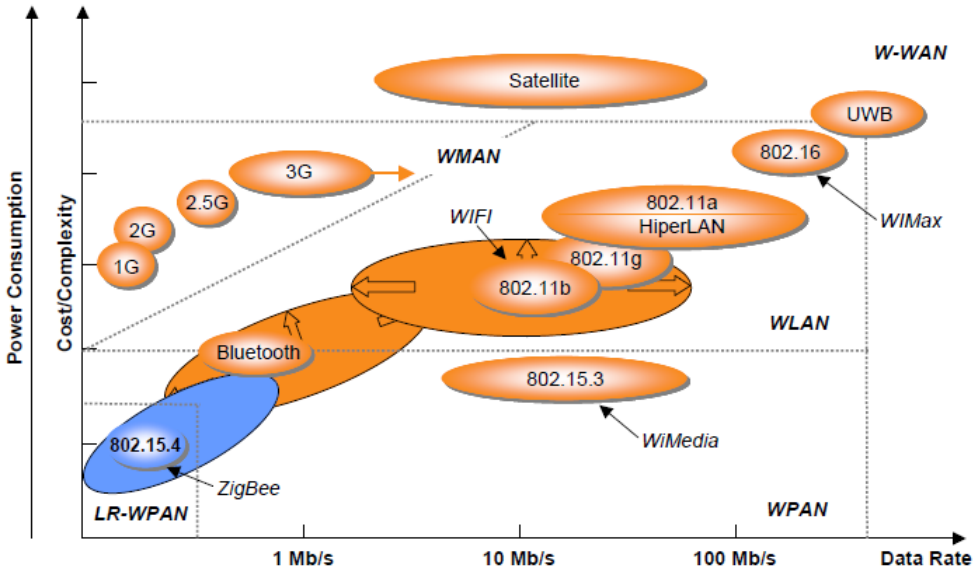



Figure 1.1: Comparison of Communication Protocols[2]

A Comparison of the existing communication standards is shown in figure 1.1, where IEEE 802.15.4 and Zigbee fulfills all the requirements of the MetroNet system.

## 1.1 IEEE 802.15.4 Standard

IEEE 802.15.4 is a standard which specifies the physical layer and media access control for low-rate wireless personal area networks (LR-WPANs). In contrast to other approaches which offer more bandwidth and require more power, emphasis is on very low cost communication of nearby devices with little to no underlying infrastructure and low power consumption[2],[3],[4].

Its basic framework allows a 10-meter communication range with a transfer rate of 250 kbps. The main identifying feature of IEEE 802.15.4 among WPANs is the extremely low manufacturing and operational costs and technological simplicity.

### 1.1.1 PHY Layer

The PHY layer is the closest layer to hardware and directly controls and communicates with the radio transceiver. It not only defines the minimum hardware level requirements but also specifies the PHY protocol functions and interactions with the MAC layer. The PHY layer is responsible for activating and deactivating the radio transceiver, and

transmitting and receiving data. It selects the channel frequency, the exact frequency at which the transceiver will operate. The frequency channels are defined through a combination of channel numbers and channel pages as shown in the Table 1.1.

Table 1.1: Channel Assignment in IEEE 802.15.4 standard

| Channel Page | Channel Number | Description |
|---|---|---|
| 0 | 0 | 868 MHz band, Europe, BPSK |
| | 1-10 | 915 MHz band, America, BPSK |
| | 11-26 | 2.4 GHz band, Worldwide, O-QPSK |
| 1 | 0 | 868 MHz band, ASK |
| | 1-10 | 915 MHz band, ASK |
| | 11-26 | Reserved |
| 2 | 0 | 868 MHz band, O-QPSK |
| | 1-10 | 915 MHz band, O-QPSK |
| | 11-26 | Reserved |
| 3-31 | Reserved | Reserved |

2.4 GHz PHY Channel assignment can be seen in Figure 1.2. Each channel is 2 MHz wide and their center frequencies are spaced 5 MHz apart. Hence, the center frequencies are calculated from the following equation:

$$CenterFrequency(MHz) = 2405 + 5 \times (ChannelNumber - 11) \tag{1.1}$$

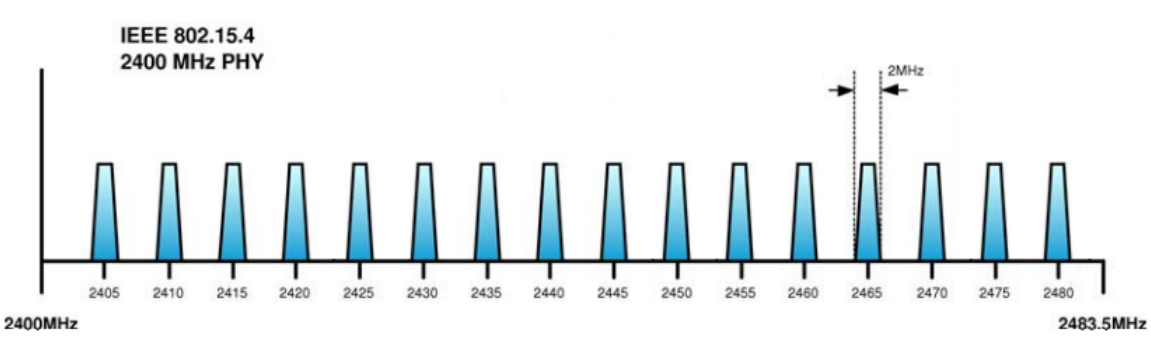



Figure 1.2: 2.4 GHz Channels[5]

The PHY layer also performs Energy Detection(ED). ED is the task of estimating the signal energy within the frequency band of interest. This estimate is used to understand whether or not a channel is clear and can be used for transmission. Carrier Sense(CS) is a verification of availability of the frequency channel. In contrast to ED, in CS the signal is demodulated to know whether the occupying signal is compliant to the IEEE 802.15.4 PHY. PHY also performs Clear Channel Assessment (CCA) on

request from MAC layer, to ensure that the channel is not in use by any other device. In CCA, the results of ED or CS can be used to decide whether a frequency channel should be considered available or busy. The PHY layer generates a link quality indicator (LQI) as well, which is an indication of the quality of the data packets received by the receiver. The received signal strength (RSS) can be used as a measure of the signal quality.

### 1.1.2 MAC Layer

The MAC provides the interface between the PHY and the next higher layer above the MAC. The MAC layer generates beacons, if the device is a coordinator. And synchronizes the device to the beacons in a beacon enabled network. It employs the CSMA-CA for channel access and manages GTS channel access as well. MAC layer provides PAN association and disassociation services. A much detailed explanation of IEEE 802.15.4 and MAC layer functions is in the book [3].

### 1.1.3 Top Layers

There are several protocols which use 802.15.4 as its MAC layer. Some of them are illustrated here:

- **Zigbee:** Zigbee standard defines a communication layer at level 3 and uppers in the OSI model. Its main purpose is to create a network topology to let devices communicate among them.

  ZigBee offers following services:

  - Encryption services : application and network keys implement extra AES encryption.
  - Association and authentication : only valid nodes can join the network.
  - Routing protocol : AODV, a reactive ad hoc protocol is implemented to perform the data routing and forwarding process to any node in the network.

- **Digimesh:** is the Digi's own mesh protocol where all the nodes can sleep and route their brother's packets using a variant of AODV[6].

  DigiMesh contains the following features:

- Self-healing : Any node may enter or leave the network at any time without causing the network as a whole to fail.
- Peer to peer architecture : Rather than maintaining a network map, routes will be discovered and created only when needed.
- Selective acknowledgments : Only the destination node will reply to route requests.
- Sleep Modes : Low power sleep modes with synchronized wake are supported with variable sleep and wake times.

## 1.2 AODV Routing Protocol

AODV is a pure on-demand routing protocol which bases route discovery on a route request and route reply query cycle and its metric is based on the number of hops from the source to the destination. AODV routing is explained in detail in [7].

In general terms, when a source node aims to send data to a destination node, the source broadcasts a route-request packet in order to discover a route to the destination. The intermediate nodes will forward this received route-request by broadcasting, and eventually, any node which has a route to the destination or the destination itself will reply with a route-reply message to the source through unicast.

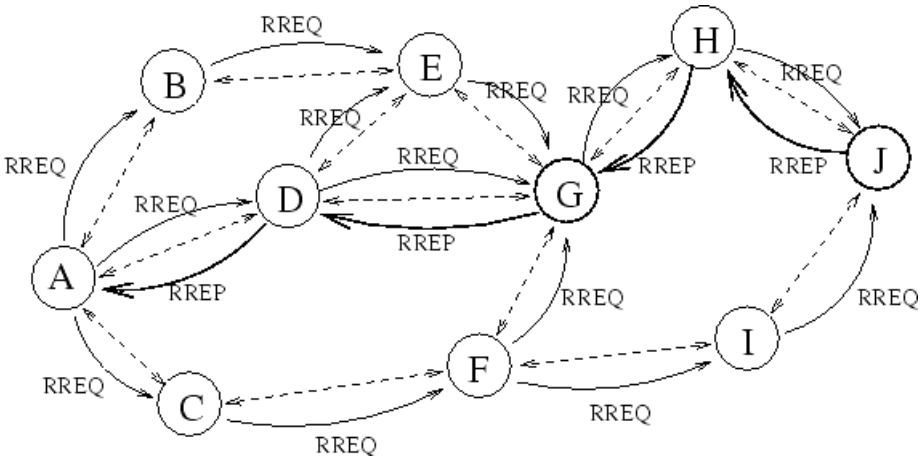



Figure 1.3: AODV routing protocol[8]

After the source receives the route-reply, it sends data to the destination. Routes are maintained and if any error occurs during the routes' lifetime, a route-error message is propagated in order to avoid the use of the broken link and out-of-date routes.

Summarizing the messages used during route discovery and maintenance processes: route request (RREQ) to discover network routes, route reply (RREP) to answer route requests, and route error (RRER) to notify link failures[9].

In the figure 1.3, node A has to send data to node J. It broadcasts the RREQ, the intermediate nodes which receive the RREQ will rebroadcast the RREQ if they do not have the route to destination. The same is repeated till the destination node J receives the RREQ and sends back RREP to the source node A through the path taken by RREQ. After having received the RREP, the node A sends data to node J through the discovered path.

## 1.3 Objective of the Project

The existing IERMON uses GSM for reporting the environmental gamma radiation level through SMS. Use of GSM adds up installation cost and maintenance cost as well, for paying SMS bills. Also, GSM being power hungry consumes about 2A current while transmitting data which increases the power consumption forcing huge solar panel placement which further increases the size. This project aims at increasing the density of IERMON network for better monitoring, using an alternate communication scheme with reduced cost, power consumption and size compared to the one existing. Also sensor data is to be communicated, at regular intervals only if a preset threshold is crossed.

## 1.4 Organization of Thesis

Chapter 1 gives a brief introduction about the project objectives and the related background required to proceed. It discusses regarding IEEE standard 802.15.4 layers and AODV routing protocol. In Chapter 2, hardware choices and the built-in routing protocol implementations are explained.

Chapter 3, describes the implementation of the developed routing protocol. The details of the tests conducted and results obtained are presented in Chapter 4. In Chapter 5, conclusion and the future scope of the project are discussed.

# CHAPTER 2

# EXISTING PROTOCOLS AND THEIR IMPLEMENTATION

## 2.1 Design Decisions

General network topologies supported in WSN are shown in figure 2.1.

- Star Topology : nodes cannot communicate directly with other nodes; all communication must be routed through the central node called the master. Each node is a slave of the central node.

- Tree Topology : nodes communicate to the central node through a fixed route. Any failure in this route, breaks the communication link between the node and central node.

- Mesh Topology: allows data to "hop" from node to node, making the network self-healing. Each node is able to communicate with the other node, as data is routed from node to node until it reaches the desired location.

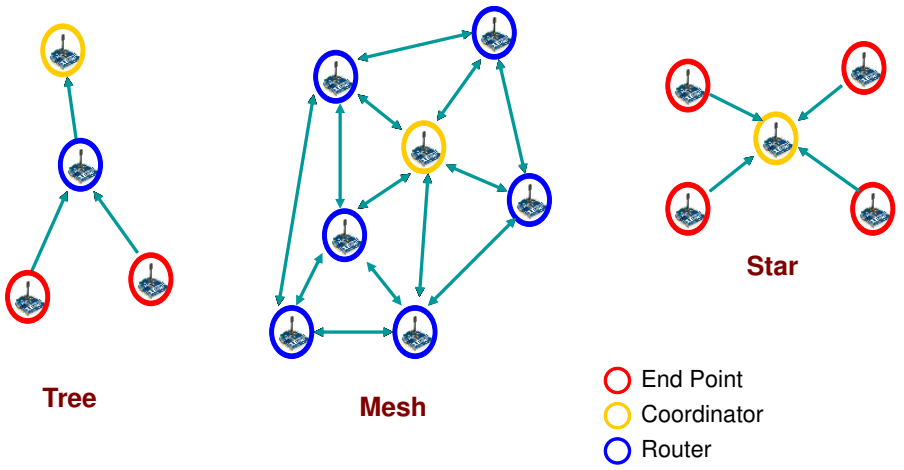



Figure 2.1: Network Topologies

Star and Tree topologies provide only one path for transmission of data to the central node, which is the data aggregator. If that particular link fails due to some reason, the

communication between the nodes die and hence is not reliable. In contrast to this, mesh networking is a powerful way to route data. Range is extended by allowing data to hop from node to node and reliability is increased by self healing. The ability to create alternate paths when one node fails or a connection is lost, makes mesh topology more reliable and robust.

The WSN using LR-WPAN devices needs such a robust routing protocol as these devices have limited communication range. To extend the range, mesh network is the most suitable. Hence while choosing IEEE 802.15.4 based modules for monitoring applications, opting the mesh network will be apt.

Among all the products available in market which support mesh routing, XBee modules were found to be more convenient to use because of its ease of practical implementation, small size, low cost and low power consumption.

ZigBee is one of the popular mesh networking protocol, which is specifically designed for low-data rate, low-power applications and uses AODV routing protocol. Digi International offers several products based on IEEE 802.15.4, XBee ZB S2 modules are one of them which provides out-of-box zigbee implementation with minimum configuration and application layer programming.


Figure 2.2: XBee ZB S2 module

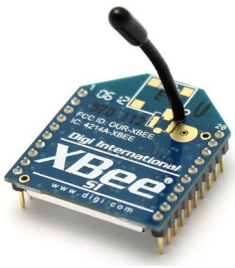



Figure 2.3: XBee S1 module

Digimesh is Digi's proprietary alternate mesh protocol. Digimesh also uses AODV for routing. XBee S1 modules with basic IEEE 802.15.4 layers along with Digimesh firmware and application layer programming have been used for Digimesh implementation.

Before describing implementation of Zigbee and Digimesh using the XBee modules, general operation of XBee RF modules should be known.

The following are the main components of the XBee Module:

- MC9S08GT60 microcontroller

- MC13193 RF chip

- RF switch that switches the antenna between Transmit and Receive

Though, Xbee modules have in-built microcontroller, programming it is not allowed as it might alter the basic functioning of the zigbee stack reciding in it. Hence, in order to have additional features, use of an external microcontroller is necessary. For this purpose Arduino Uno board with ATmega328 is used. XBee Shield is used as an interface between arduino board and XBee as shown in figure 2.4.

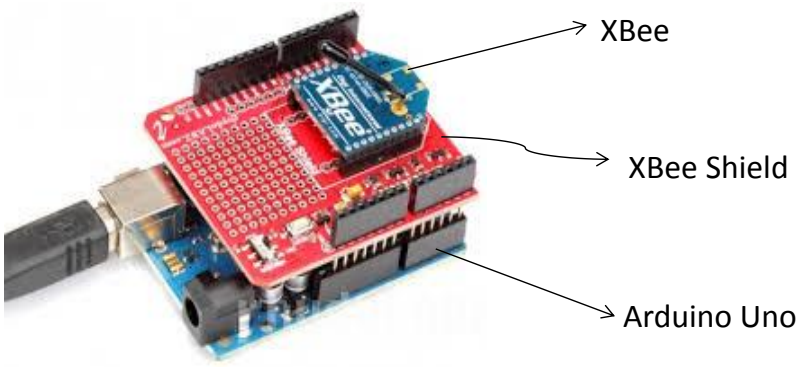



Figure 2.4: XBee and Arduino Uno interface using an XBee Shield

Like any other WSN, each sensor node has RF module (XBee here), external microcontroller unit along with peripherals (Arduino Uno here), XBee Shield for their interface, Sensor unit (temperature senor or potentiometer) and a rechargeable battery for power supply. The basic block diagram of the sensor node in prototype MetroNet is as shown in figure 2.5.

Another reason for choosing an external microcontroller is the UART usage of XBee modules. Data sent to XBee through UART will get transmitted, and the data received by XBee can be read by the external microcontroller easily through the UART.

Also, the XBee modules support both transparent and API (Application Programming Interface) serial interfaces.

- Transparent operation : When a module operates in transparent mode, it acts as a

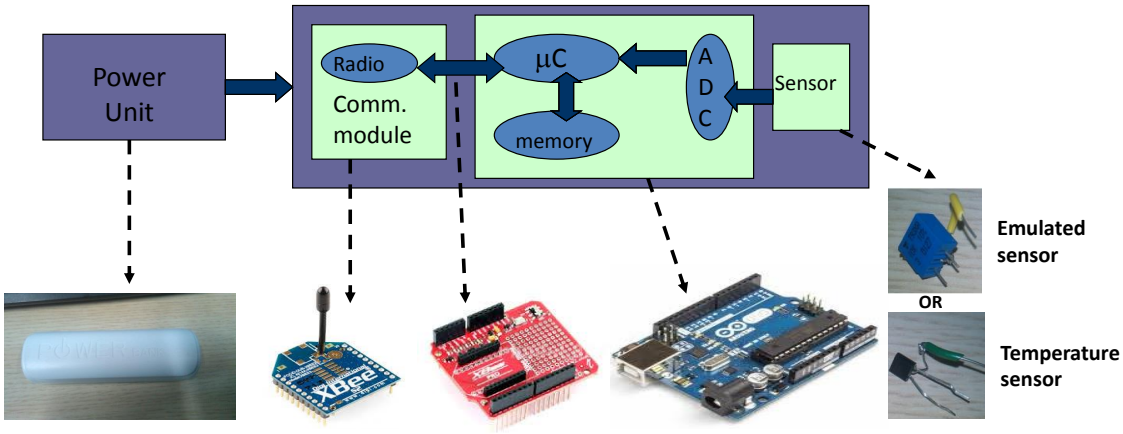



Figure 2.5: Basic Block Diagram of a Wireless Sensor Node

serial line replacement. All the UART data received through the DIN pin of XBee is queued up for RF transmission. When a module receives RF data, it sends the data out through its DOUT pin.

- API operation : API operation is an alternative to transparent operation. The frame-based API extends the level to which a host application can interact with the networking capabilities of the module. When you operate the device in API mode, all data entering and leaving the UART is in the form of frames, that define operations or events within the module. API frame format is as shown in figure 2.6

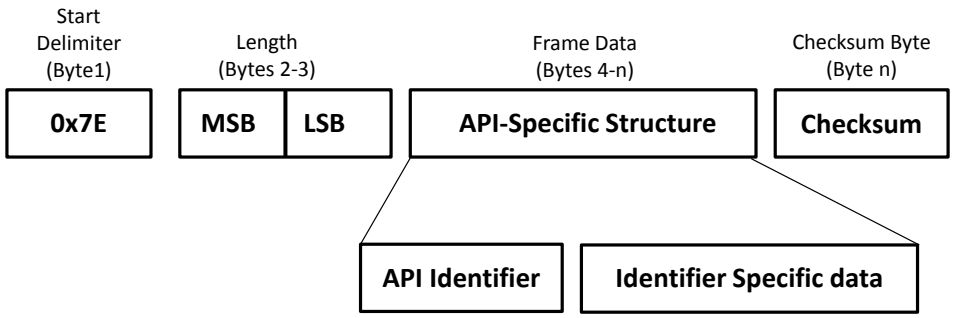



Figure 2.6: API frame format

All the XBee modules must be set exactly to the same following configurations : 8 bits of data, no parity bit, 1 stop bit, at 9600 baud (8N1 @ 9600 baud).

## 2.2 Zigbee

Zigbee is implemented using XBee ZB S2 modules. Some of the XBee ZB S2 specifications are mentioned in Table 2.1.The basic configurations of these modules are set using X-CTU software.

Table 2.1: XBee ZB S2 Specifications

| **Parameter** | **Value** |
|---|---|
| Range | indoor: 40m<br>outdoor: 120m |
| Operating Current (Transmit) | 45mA |
| Operating Current (Receive) | 45mA |
| Supported Topology | Star, Tree, and Mesh |

The ZigBee Protocol defines three types of nodes: Coordinators, Routers and End Device, with a requirement of at least one Coordinator per network. While all nodes can send and receive data, there are differences in the specific roles they play.

- Coordinators: There is exactly one coordinator in each network. It initiates the Zigbee network and can route data as well.
- Routers: acts as intermediate nodes, relaying data from other devices.
- End Device: can be low-power / battery-powered. They have sufficient functionality to talk to their parents (either the coordinator or a router) and cannot relay data from other devices.

However, in MetroNet, to ensure scalability of network end-devices are not used. Instead, a zigbee network only with coordinator and routers is formed for future network extension, if required.

All ZigBee devices have two different addresses, a 64-bit and a 16-bit address.

- 64-bit address : is a unique device address assigned during manufacturing. The 64-bit address is also called the extended address.
- 16-bit address : device receives a non-unique 16-bit address when it joins the network. The 16-bit address of 0x0000 is reserved for the coordinator

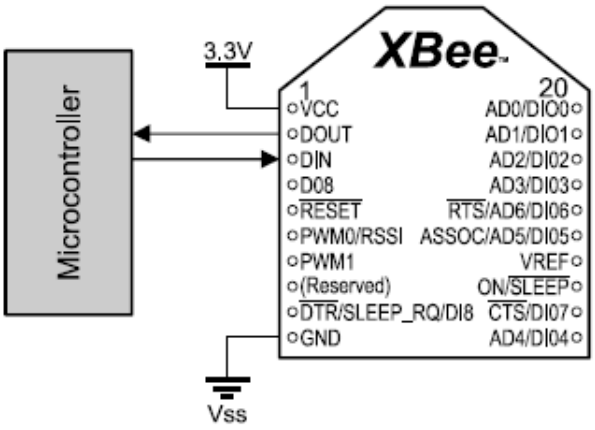



Figure 2.7: XBee pin configuration[10]

### 2.2.1 Implementation

First and foremost the XBee ZB S2 modules are configured using the X-CTU software. All the devices in the network should have same PAN-ID and should be operating in same channel.

Coordinator uses API mode of operation whereas Routers use AT mode. Coordinator is operated in API as the receive packet frames would constitute both the data and the originators device address, making it easier to know which router/ sensor node has transmitted that particular sensor value. Routers need not operate in API as it has to just send the sensor value if it has crossed the preset threshold, hence AT mode was found sufficient.

#### X-CTU configuration

The X-CTU configuration for coordinator XBee ZB S2 are as follows:

- PAN ID(**ID**): 0x88
- Scan Channels(**SC**): 0x0C
- Destination Address High(**DH**): 0
- Destination Address Low(**DL**): 0xFFFF
- Node Identifier(**NI**): C0

- Broadcast Hops(**BH**): 0

- Power Level(**PL**): 0x04

The coordinator should be able to communicate to all routers and hence its destination address is kept as the broadcast-address (0xFFFF). Node ID is a user defined name given to the device, for coordinator 'C0' is the NI. Broadcast Hops is set to 0, to allow maximum hops and the output power level is set to highest i.e, 4 for maximum range of communication.

The X-CTU configuration for router XBee ZB S2 are as follows:

- PAN ID(**ID**): 0x88

- Scan Channels(**SC**): 0x0C

- Destination Address High(**DH**): 0

- Destination Address Low(**DL**): 0

- Node Identifier(**NI**): R#

- Broadcast Hops(**BH**): 0

- Power Level(**PL**): 0x04

The routers should send all its sensor values to the coordinator and hence their destination address is set to 0 which is the default network-address of the coordinator in the network. Node ID is set to 'R#' i.e, R followed by a number for the routers.

## 2.3 Digimesh

Digimesh is Digi's propitiatory mesh protocol. XBee S1 modules have been used to implement Digimesh.Few XBee S1 specifications are mentioned in Table 2.2.

DigiMesh has only one node type. As a homogeneous network, all nodes are capable of routing data and are interchangeable. All of them can operate as battery-powered devices.

Table 2.2: XBee S1 Specifications

| Parameter | Value |
|---|---|
| Range | indoor: 30m<br>outdoor: 100m |
| Operating Current (Transmit) | 45mA |
| Operating Current (Receive) | 55mA |
| Supported Topology | Star, Tree, and Mesh(with Digimesh) |

XBee S1 modules have only a device address or the 64-bit address. In API-frames of XBee S1, the network address field is filled with 0xFFFE which implies reserved.

### 2.3.1 Sleep Modes in Digimesh

Digimesh firmware provides additional sleep modes for an enhanced power saving capability. The sleep modes supported by Digimesh are as follows:

- Asynchronous pin sleep mode : When pin 9 is asserted (high), the module will enter a low-power state. The module wakes from pin sleep when the pin 9 is de-asserted (low).

- Asynchronous cyclic sleep mode : XBee module sleeps cyclically, i.e, Cyclic sleep allows the module to sleep for a specified time and wake for a short time to poll.

- Asynchronous cyclic sleep with pin wake up mode : This is the same as regular cyclic sleep mode but with the option of also waking the module using physical pin 9.

- Synchronous sleep support mode : A node with synchronous sleep support mode will synchronize itself with a sleeping network, but will not sleep itself. A sleep support node transmits data only when the other nodes in the sleeping network are awake.

- Synchronous cyclic sleep mode: A node with synchronous cyclic sleep mode, sleeps for a programmed time, wakes in unison with other nodes, exchanges data and sync messages, and then returns back to sleep.

### 2.3.2 Implementation

The XBee S1 modules are configured using X-CTU software and the arduino uno boards are programmed. The coordinator is API enabled. The Synchronous cyclic sleep mode is used in all devices in network for low-power operation. All the devices are maintained in a common PAN-ID of '888' and channel 'C' is used, also all the modules are loaded with Digimesh firmware using X-CTU software.

#### X-CTU configuration

The X-CTU configuration for coordinator XBee S1 are as follows:

- PAN ID(**ID**): 0x888
- Scan Channels(**SC**): 0x0C
- Destination Address High(**DH**): 0
- Destination Address Low(**DL**): 0xFFFF
- Node Identifier(**NI**): C0
- API enable(**AP**): 1
- Sleep Mode(**SM**): 8(Synchronized Cyclic Sleep)
- Sleep Options(**SO**): 1(Sleep-Coordinator)
- Sleep Time(**SP**):0x1194
- Wake Time(**ST**): 0x3A98
- Broadcast Hops(**BH**): 0
- Power Level(**PL**): 0x04

The broadcast address is kept as the destination address, for data aggregator. All the modules are sleep enabled with synchronized cyclic sleep mode and the data aggregator module is made the sleep coordinator implying that it decides the time for which the entire network sleeps and wakes.

Sleep Time(SP) is set to 0x1194 which is hex value for 4500, according to the configurations the network will sleep for:

$$sleep\ time = SP(in\ decimal) \times 10\, ms \tag{2.1}$$

Hence here the network will sleep for 45 seconds.

Wake Time(ST) is set to 0x3A98 which is hex value for 15000, according to the configurations the network will stay awake for:

$$wake\ time = ST(in\ decimal)\, ms \tag{2.2}$$

Hence here the network will be awake for 15 seconds. The sleep coordinator forces the rest of the network to sleep according to the set time by broadcasting sync messages.

The following are the X-CTU configurations required for a digimesh router:

- PAN ID(**ID**): 0x888
- Scan Channels(**SC**): 0x0C
- Destination Address High(**DH**): 0
- Destination Address Low(**DL**): 0
- Node Identifier(**NI**): R#
- API enable(**AP**): 0
- Sleep Mode(**SM**): 8(Synchronized Cyclic Sleep)
- Sleep Time(**SP**):0x1F4
- Wake Time(**ST**): 0xD6D8
- Broadcast Hops(**BH**): 0
- Power Level(**PL**): 0x04

The routers should send all its sensor values to the sleep coordinator i.e, the data aggregator and hence their destination address is set to '13A20040A89439' which is the

64-bit address of the sleep coordinator. Node ID is set to 'R#' i.e, R followed by number given to router.

SP and ST in routers are the sleep and wake times of the modules before they sleep in sync with the sleep coordinator. Hence, its better to have the router modules awake for more time so that they get in sync easily.

Here, Sleep Time(SP) is set to 0x1F4 which is hex value for 500, according to the configurations the network will sleep for 5 sec.
Wake Time(ST) is set to 0xD6D8 which is hex value for 55000, according to the configurations the network will sleep for will be awake for 55 sec.

However, after they get in sync with the sleep coordinator, their sleep- wake cycle will be as configured in the data aggregator.

## 2.4 Algorithm

Arduino Uno for the coordinator and routers is programmed separately, a each one serves different purposes in the network.

**Router**

Router has a sensing module connected to it. Sensor data is given to an analog pin,for reading which micro-controller's 10-bit ADC is used. Program should retrieve back the original sensed data, compare it with a threshold and send it to the UART, only if the threshold is crossed. Hence, the data is transmitted only if a preset threshold is crossed. Also, micro-controller should be kept in sleep mode in such a way that ADC and UART still work while disabling others, for reduced power consumption.

**Coordinator**

Coordinator is the data collector i.e, all the routers forward the sensed data to the coordinator. Coordinator is programmed to be in API mode to ensure use of API frames and easy decoding of data and the address of the XBee generating it. Micro-controller program, reads API frame from UART, separates 64 bit MAC Address of the XBee and

the sensed data. After decoding both, it prints the same.

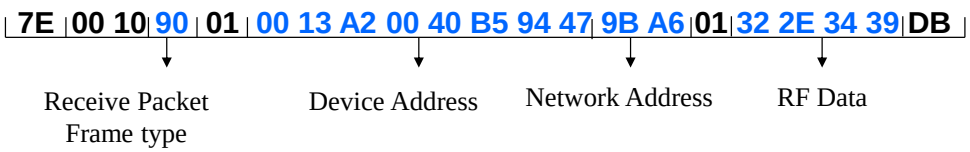



Figure 2.8: Receive Packet Frame

The Receive packet frame used in coordinator programming for extracting the originating device address and the corresponding sensor value is as shown in figure 2.8.

The frame type of ’90’ indicates that the frame is a Receive Packet, ’00 13 A2 00 40 B5 94 47’ is the 64 bit address of the originating XBee and ’32 2E 34 39’ implies that a voltage of 2.49 volts is the sensor data.

A field test for ZigBee was performed at IIT Madras Stadium with about 14 nodes wherein the WSN was found to work according to the need.

## 2.5 Conclusion

Sensor data was transmitted at regular intervals only when it crosses a preset threshold. This was accomplished using both ZigBee and DigiMesh. ZigBee implementation has lesser power consumption when compared to the existing GSM. DigiMesh incorporates sleep mode operation and further reduces the power consumption.

# CHAPTER 3

# ROUTING ALGORITHM DEVELOPMENT

The implementation of prototype MetroNet with the existing protocols like zigbee and digimesh serves the purpose of communicating the sensed data to the central data collector. The application requires sensor data to be communicated only if the threshold is crossed, hence when a node dies, the nodes which were relying on the presently non-active node, to parse its data to the coordinator will choose another route. Therefore, in a case of a node failure, it would be left unnoticed in the implemented protocols and the sensor value in the area where that particular non-active node was deployed remains unmonitored.

To avoid such a situation two approaches can be adopted.

- In one of which, every node whether it crosses the preset threshold or not should communicate its sensed value to the coordinator. However, the nodes that cross the threshold send their sensed data more frequently. Say for example, the node with threshold crossed transmits its data every 1 hour and the other nodes transmit for every 5 hours.
- Other way is knowing and assessing the path taken by the sensed data to notice the node failures and link failures in a much efficient way. However, in the implemented protocols knowing the path is not possible and hence, a routing algorithm to serve this purpose has been developed. This routing algorithm also includes supplied battery voltage level as one of the routing parameters.

API provides means of configuring the nodes and routing data at the application layer. A host application can send data frames with the destination address and payload information decided in run-time, instead of using command mode to modify addresses. Using the same, a routing algorithm in the host application layer has been developed.

In this routing algorithm, each node has information of only the neighboring nodes. Neighboring nodes here imply the nodes which are in the immediate range of communication. This algorithm determines the best next-hop node among the neighboring

nodes, and each node forwards the data to the coordinator through their best next-hop node.

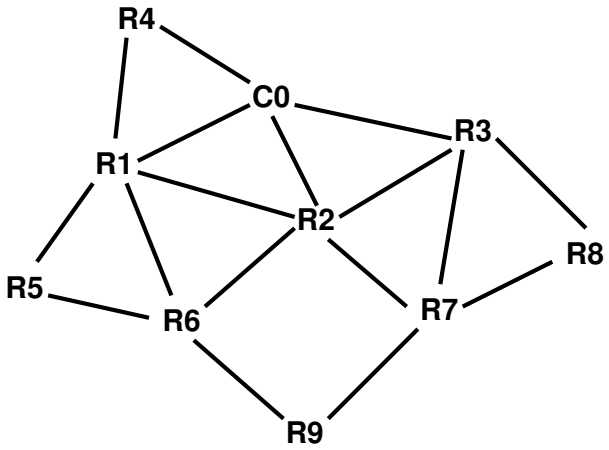



Figure 3.1: Sample Network

Consider a network as shown in Figure 3.1. The nodes are scattered, the coordinator is denoted as C0 and the rest of the 9 routers nodes are denoted as R1, R2 etc. The nodes which are joined with the edges are neighboring nodes and can communicate directly without any need of a router node. However, the nodes with no connecting edge in between, needs the help of one of its neighboring nodes to talk to the non-neighboring nodes.

## 3.1 Node Discovery

First step of the algorithm is Node Discovery. In this, each node discovers its neighboring nodes, and stores them in neighbor list (NL) and their addresses in node table. Before performing its node discovery, coordinator instructs all the routers of the network to start node discovery process. The Node Discovery process of the coordinator is as shown in the Figure 3.2. In the Sample Network, the nodes R1, R2, R3, and R4 are the neighbors of the coordinator. Hence, after Node Discovery, coordinator will have R1, R2, R3,and R4 in its Neighbor List.

In a similar way all the nodes along with the coordinator performs node discovery simultaneously3.3, and by the end of the Node Discovery Process all the nodes acquire their corresponding Neighbor Lists as in Table 3.1.

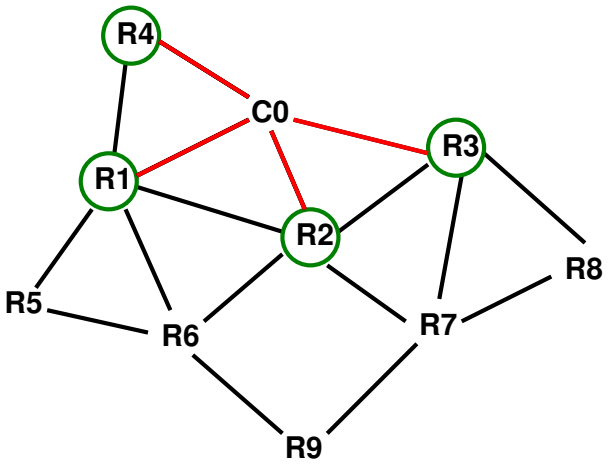



Figure 3.2: Node Discovery by Coordinator (NL for coordinator(C0): R1 R2 R3 R4)

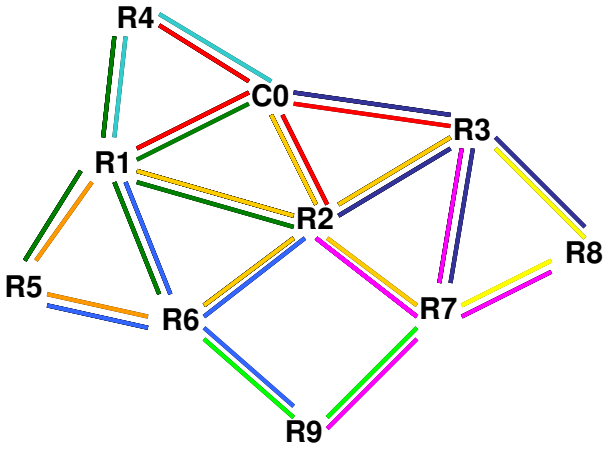



Figure 3.3: Node Discovery Process(Each color indicate the neighbor node discovery performed by each node)

Table 3.1: Neighbor Lists

| | | | | | |
|---|---|---|---|---|---|
| C0 | R1 | R2 | R3 | R4 | |
| R1 | C0 | R2 | R4 | R5 | R6 |
| R2 | R1 | C0 | R3 | R6 | R7 |
| R3 | R2 | C0 | R8 | R7 | |
| R4 | R1 | C0 | | | |
| R5 | R1 | R6 | | | |
| R6 | R1 | R2 | R5 | R9 | |
| R7 | R2 | R3 | R8 | R9 | |
| R8 | R3 | R7 | | | |
| R9 | R6 | R7 | | | |

## 3.2 Determining the Next Hop Nodes

After the Node Discovery process, every node determines the next-hop nodes. The next hop nodes of a node will be the neighboring nodes itself, but before transmitting data to coordinator through a neighboring node its necessary that the neighboring node chosen to parse the data should take minimum number of hops to reach the coordinator. For this, all the neighboring nodes are sorted accordingly into a next hop node list. To sort, following procedure is used.

### 3.2.1 Transmission of Neighbor List by Coordinator and its comparison

The coordinator broadcasts its neighbor list to all its neighbor nodes. These neighboring nodes have a neighbor list of their own. Each node compares its own neighbor list with the coordinator's neighbor list.

### 3.2.2 Node Classification

The neighboring nodes are classified into Parent, Sibling, and Child nodes. Using this classification, the nodes are sorted in Next Hop node list. The node from which the neighbor list has been received will be the parent node. Each node's neighbor is compared to the parent's neighbor list and if a match is found then the node is a Sibling node. If a match is not found then the node is a Child node.

For example, Node R2 has a NL of R1,R3,R6,R7 and it receives Coordinator's NL: R1,R2,R3,R4. Therefore, C0 will be R1's Parent node. R1 and R3 being present in coordinator's NL also, makes them the Sibling nodes. And the rest of the nodes i.e, R6 and R7 will be its Child nodes.

Therefore, for R2

Parent node : C0

Sibling node: R1 R3

Child node : R6 R7

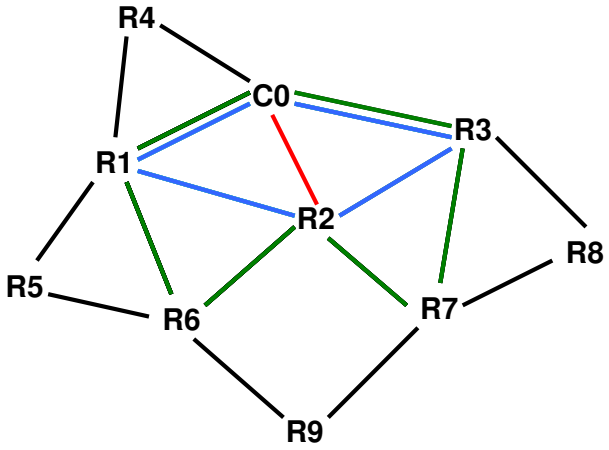



Figure 3.4: Node Classification of node R2 (red link indicate data transmission to parent node, blue indicate sibling, and green indicate child node)

The same procedure of node classification is followed by all the coordinator's neighbor nodes. After the node classification, Next hop nodes have to found. If the data is traversing through parent node, it reaches the coordinator with minimum hops.

As it can be seen in Figure 3.4, some of the paths through which data can be sent from R2 to C0 are:

| | | |
|---|---|---|
| R2-C0 | 1 hop | parent node |
| R2-R1-C0 | 2 hops | sibling node |
| R2-R3-C0 | 2 hops | sibling node |
| R2-R6-R1-C0 | 3 hops | child node |
| R2-R7-R3-C0 | 3 hops | child node |

Its clear from the Figure 3.4 that data takes minimum hops if traversing through parent. Also, Sibling nodes will be preferably taking less hops when compared to the Child Nodes. Hence, parent nodes are placed first in next hop node list, then the sibling nodes followed by the child nodes.

This node classification process in R1, R2, R3, and R4 gives the result as shown in Table 3.2.

Table 3.2: Node Classification in coordinator's neighbors

R1

| Parent Node | C0 | | | | |
|---|---|---|---|---|---|
| Sibling Node | R2 | R4 | | | |
| Child Node | R6 | R5 | | | |
| Next Hop Node | C0 | R2 | R4 | R6 | R5 |

R3

| Parent Node | C0 | | | |
|---|---|---|---|---|
| Sibling Node | R2 | | | |
| Child Node | R7 | R8 | | |
| Next Hop Node | C0 | R2 | R7 | R8 |

R2

| Parent Node | C0 | | | | |
|---|---|---|---|---|---|
| Sibling Node | R1 | R3 | | | |
| Child Node | R6 | R7 | | | |
| Next Hop Node | C0 | R1 | R3 | R6 | R7 |

R4

| Parent Node | C0 | |
|---|---|---|
| Sibling Node | R1 | |
| Child Node | | |
| Next Hop Node | C0 | R1 |

### 3.2.3 Transmission of Neighbor List to child nodes

After Node classification, coordinator's neighbor nodes sends their own neighbor lists to their Child nodes. For example, R1 will send its neighbor list to its child nodes R5 and R6. Then the child nodes also perform the same node classification process by comparing its node list with its parents node list. By the end, when all nodes have received their parent's neighbor list and have performed the node classification, the result would be as shown in Table 3.3

Table 3.3: Node Classification

R6

| Parent Node | R1 | R2 | | |
|---|---|---|---|---|
| Sibling Node | R5 | | | |
| Child Node | R9 | | | |
| Next Hop Node | R1 | R2 | R5 | R9 |

R7

| Parent Node | R2 | R3 | | |
|---|---|---|---|---|
| Sibling Node | R8 | | | |
| Child Node | R9 | | | |
| Next Hop Node | R2 | R3 | R8 | R9 |

R5

| Parent Node | R1 | |
|---|---|---|
| Sibling Node | R6 | |
| Child Node | | |
| Next Hop Node | R1 | R6 |

R8

| Parent Node | R3 | |
|---|---|---|
| Sibling Node | R7 | |
| Child Node | | |
| Next Hop Node | R3 | R7 |

R9

| Parent Node | R6 | R7 |
|---|---|---|
| Sibling Node | | |
| Child Node | | |
| Next Hop Node | R6 | R7 |

### 3.2.4 Data Transmission and Acknowledgment Forwarding

Data is to be transmitted by the nodes only if they cross the preset threshold or they receive data from some other node. If a node crosses the threshold, then it transmits its data to its next hop nodes after assessing the next hop nodes' battery voltage level and waits for an acknowledgment from the coordinator. If it does no receive the acknowledgment or if the parents battery voltage was not sufficient enough then it transmits the data to the next node in Next Hop node list. If the node receives data from some node, it checks whether its own threshold is crossed or not. It appends its data to the received data, in case where the threshold has been crossed. Else, it appends the data in such a way that the coordinator knows that the data has parsed through this particular node. Also the acknowledgment received from the coordinator will be forwarded by the intermediate nodes to the node which has requested for data transfer.

## 3.3 Prototype Implementation

All the XBee modules are configured to be in API mode to enable dynamic change in their destination address. XBee ZB S2 modules have been used for this implementation.

#### X-CTU configuration

The X-CTU configuration for coordinator XBee ZB S2 are as follows:

- PAN ID(**ID**): 0x88
- Scan Channels(**SC**): 0x0C
- Destination Address High(**DH**): 0
- Destination Address Low(**DL**): 0xFFFF
- Node Identifier(**NI**): C0
- Node Discovery Options(**NO**): 3
- Broadcast Hops(**BH**): 0

- Power Level(**PL**): 0x04

- AD1/DIO1 Configuration(**D1**): 2

Most of the configurations for both coordinator and router are similar to that in regular zigbee implementation. Some additional configurations include, Node Discovery Option which is set to 3 implying that the node returns its own node discovery as well. The AD1/DIO1 is set to 2 which identifies XBee pin 19 to be a Analog input pin. It is this pin through which battery voltage assessment is done.

The X-CTU configuration for router XBee ZB S2 are as follows:

- PAN ID(**ID**): 0x88

- Scan Channels(**SC**): 0x0C

- Destination Address High(**DH**): 0

- Destination Address Low(**DL**): 0

- Node Identifier(**NI**): R#

- Node Discovery Options(**NO**): 3

- Broadcast Hops(**BH**): 0

- Power Level(**PL**): 0x04

- AD1/DIO1 Configuration(**D1**): 2

For ease of understanding, consider the network shown in figure 3.5 for implementation.

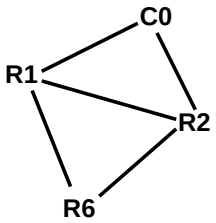



Figure 3.5: Example Network for Implementation

Coordinator broadcasts the begin frame as in figure 3.6, instructing all the routers in the network to start performing the node discovery process. '10' in the frame indicate

that the frame is a transmit request frame, ’00 00 00 00 00 00 FF FF’ and ’FF FC’ are the broadcast 64-bit and 16-bit addresses.

Broadcast Hop(BH) is ’00’ as show in figure to ensure that all the routers receive the begin frame. ’42’ which is the hexadecimal value for ’B’ indicating that it is the begin frame.

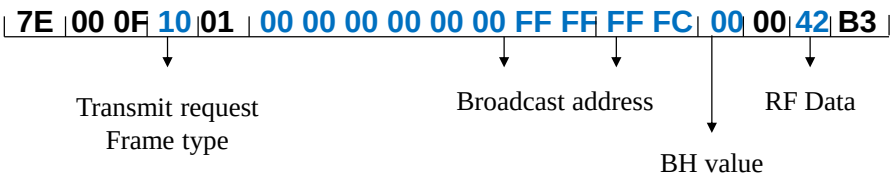



Figure 3.6: Begin frame broadcast by the coordinator

All the nodes perform Node Discovery simultaneously. For the nodes to know information about their neighboring nodes only, BH parameter is set to 1 using a local AT Command. The API frame for the BH-AT Command is as shown in Figure 3.7. ’08’ frame type identifies it to be AT command frame.

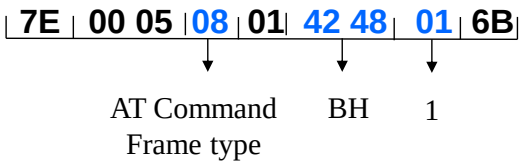



Figure 3.7: BH AT command

’42 48’ in the frame indicate the AT command BH. ’01’ sets the BH to 1.

To discover all the existing neighboring nodes, ATND command is locally sent to each XBee module. This command discovers and reports all RF modules found within the range. The command format is as shown in Figure 3.8. ’4E 44’ in the frame indicate the AT command ND.

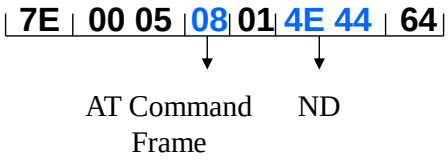



Figure 3.8: ND AT command

Sending the above frame locally results in a Node Discovery Response Frame from each of the neighboring nodes and from itself, the first response being its own. The Node Discovery response frame is a AT Response frame with frame type 0x88. It consists of the device's address, network address, node identifier string (NI) and device type.

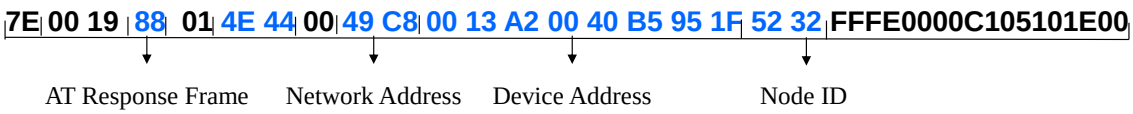



Figure 3.9: Node Discovery Response

In Figure 3.9, frame type of '88' indicates that it is a AT response frame, '4E 44' which is hexadecimal value of 'N D' indicates that it is a Node Discovery response frame. As indicated, the Node Discovery response frame consists of Device address which is the 64-bit address which is '00 13 A2 00 40 B5 95 1F' in the Figure 3.9, Network address is '49 C8' and NI being '52 32' having the ASCII value of 'R 2'. Similar frames are received from all the neighbor devices. Hence there will be as many such frames received by the node as many neighbors it has. The NI, network address, and device address are extracted from the frame and stored in nodeTable [ ][ ] array as shown in Figure 3.10. The first 2 characters indicate NI, next 2 bytes are the network address and the rest of the bytes is the device address.

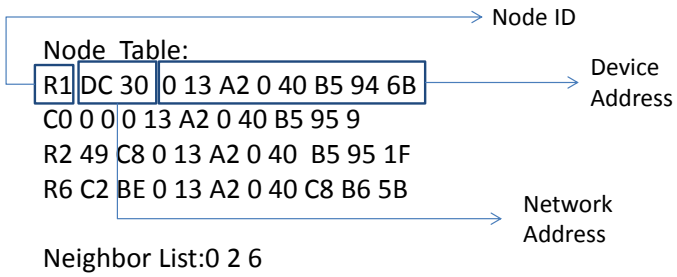



Figure 3.10: Node Table and Neighbor List

R1 being the node where the node discovery has been done, and C0, R2 and R6 being the neighboring nodes. As it can be seen in Figure 3.10, the 2nd character of the NI's of all these neighbors are stored in Neighbor List (NL).

After Node Discovery by all the nodes, the coordinator broadcasts its neighbor list to all its neighboring nodes using the frame shown in Figure 3.11. The frame is transmit

request frame having 64-bit broadcast address '00 00 00 00 00 00 FF FF' and 'FF FC' as 16-bit address. The RF data in the packet '53 00 1 2' indicates the sender of the frame i.e, source (S) being '00' which is 2nd character of the NI of node C0, and R1 R2 is the neighbor list of C0. The neighbor list transmitting frame's RF data holds 'S' followed by NI of sender and its neighbor list.

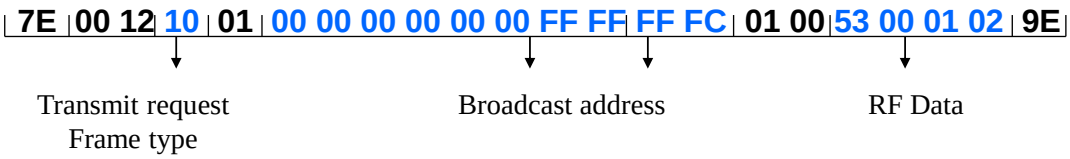



Figure 3.11: Neighbor List frame Transmitted by Coordinator

The above transmitted frame is received by the coordinator's neighboring nodes in the frame format as shown in Figure 3.12 and the node classification process as explained in 3.2.2 is performed. The '90' frame type indicates that it is a Receive Packet. '00 13 A2 00 40 B5 95 09' and '00 00' indicates the 64-bit device address and the 16-bit network address of the coordinator. In a similar way all the transmit request

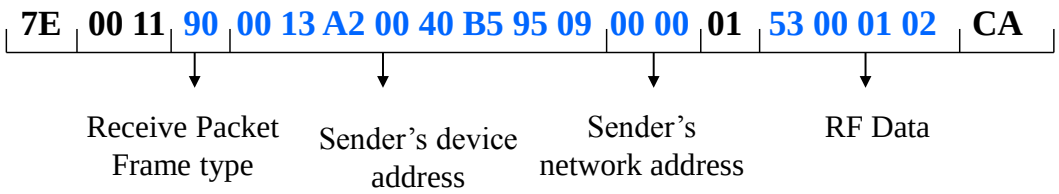



Figure 3.12: Received Packet with NL information

frames from sender are received by the destination as Receive Packet with both having same RF payload or data.

The RF data transmitted by the coordinator is received as it is and is stored in info [ ][ ]. Comparing its own neighbor list with info [ ][ ], the parent, sibling and child nodes are determined and stored in parentNode [ ], Sib [ ], and childNode [ ] respectively. After node classification, in NextHop [ ], the nodes are arranged such that first node in the array takes least number of hops to reach the coordinator. This can be seen in Figure 3.13. Each node transmits its NL to all its child nodes using a unicast rather than a

Parent Nodes:0
Child Nodes:6
Siblings: 0 2
Nest Hop Nodes:0 2 6

Figure 3.13: Node Classification

broadcast. The frames transmitted in this process are as shown in Figure 3.14 where '00 13 A2 00 40 C8 B6 5B' is the child's device address, 'C2 BE' is the 16 bit network address of R6, the child node. RF data consists of '53 01' which indicates that R1 is the source, and '00 02 06' is its neighbor list.

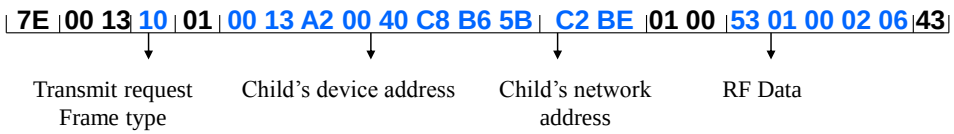



Figure 3.14: Transmitting NL information to Child Nodes

The child node performs the node classification process and transmits its own neighbor list to its child nodes. Similarly, all nodes complete the node classification process and are aware of their next hop nodes. However, here R6 has no child nodes and hence does not tramnsmit its NL to any node.

There has to be a data transmission to the coordinator only if any one of the sensor-nodes in the network crosses the preset threshold. Suppose a node has crossed its threshold, it will send 'D NI sensor data' as RF data to its next hop node after having assessed the next hop nodes' battery voltage, and wait for an acknowledgment frame to be forwarded to it, from the coordinator. The same can be seen in the Figure 3.15. In the frame, '00 13 A2 00 40 B5 94 6B' is the next hop nodes' device address, 'DC 30' is the 16 bit network address. RF data consists of '44 06 02 2E 05 01'. '44' ASCII value of 'D' indicates that it is a data frame, '06' is the source of data implying node R6 has crossed the threshold and '02 2E 05 01' is the pot voltage i.e, 2.51 volts here.

If the node sending data is coordinator's neighbor then it receives an acknowledgment from coordinator. Else, the intermediate node that receives the data forwards it to the coordinator either directly or through another router after appending the data received.

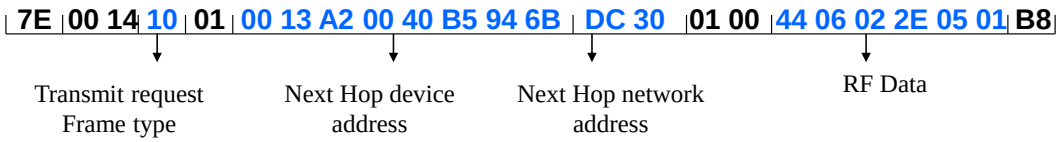



Figure 3.15: Frame for transmitting data to the next hop node

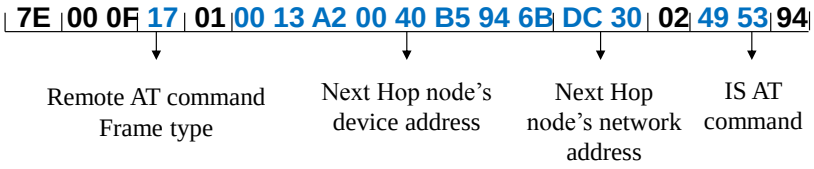



Figure 3.16: Remote AT command frame sent to next hop node to assess its battery voltage level

For next hop node's battery voltage assessment, the battery voltage of the router passed through a voltage divider, is given as an analog input to the 'AD1/DIO1' pin 19 of XBee. And when some node needs to assess the battery voltage of its next hop node to ensure whether that particular next hop will be able to relay its data, it sends a remote ATIS command frame as in figure 3.16. ATIS command is the force sample AT command which forces a read of all the enabled analog and digital pins.

In the frame '17' imply that its a remote AT command, '49 53' is hex value for 'I S'. And this frame is addressed to the next hop node whose battery voltage is being assessed. The transmission of this frame leads to a reply from the corresponding next hop node as shown in the figure3.18.

Battery voltage can be assessed using $V_b$ value which is force read using remote ATIS command, as follows:

$$V_{battery} = V_b \times (1 + R_1/R_2) \tag{3.1}$$

'97' identifier saying the frame is a remote AT response of IS denoted by '49 53'. The RF payload holds the sender's battery voltage level. RF data in this frame being '01 00 02 03 F1' where '01' gives the number of samples, '00 00' are digital and '02' is

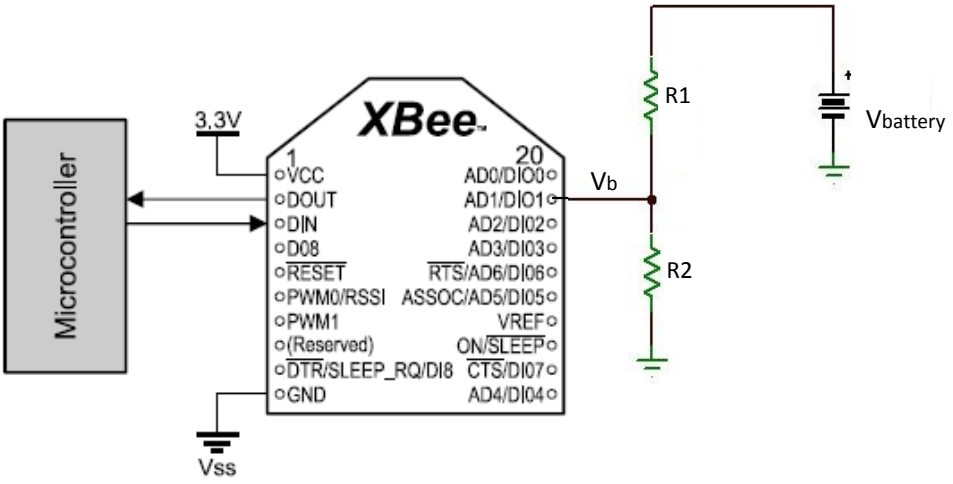



Figure 3.17: Circuit Connection to assess the battery voltage[10]

analog channel mask indicating that AD1 is enabled and,'03 F1' is ADC channel Data in hex which is the $V_b$.

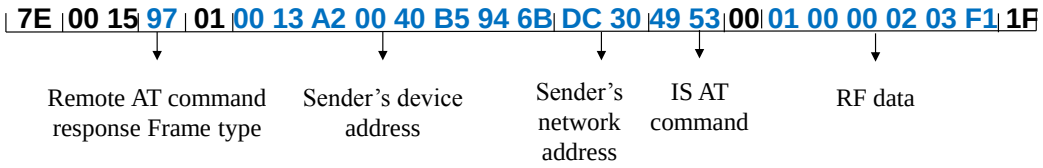



Figure 3.18: Remote AT command response frame containing battery voltage level

The actual battery voltage is fed to XBee AD1 pin 19 after having passed through a voltage divider circuit as shown in figure 3.17. This is done as XBee analog pin can take a maximum of 1.2 volts as input.

$$V_b = ((ADC\ channel\ data\ in\ decimal) \times 1200)/1024 \tag{3.2}$$

where, 1200 is the $V_{ref}$ and 1024 is $2^{10}$ as XBee has a 10-bit ADC

Hence, here ADC channel data being '03 F1' having a decimal value of '1009'. On calculating, $V_b$ is 1182.42 mV i.e, approximately 1.2 volts. Battery voltage can be found from $V_b$. If $V_b$ of the chosen next hop is found to be insufficient the data is then transmitted to the next node in the next hop list after assessing its battery voltage in a similar way.

The intermediate node checks whether its own threshold has crossed, if crossed, then it appends the received data and transmits it to its next hop node as shown in figure

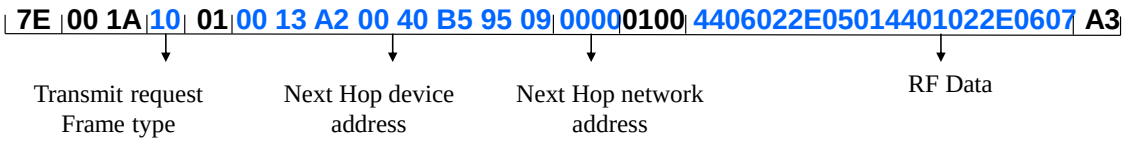



Figure 3.19: Router appending the Received data with its own data

3.19. In the frame, RF data consists of the received data followed by 'D NI sensor data' which here is '44 01 02 2E 06 07' i.e, data 'D' generated by node R1 is 2.67 volts.

If the intermediate node has not crossed the preset threshold then it simply appends the received data with 'P NI' and forwards it to its next hop after battery voltage assessment.

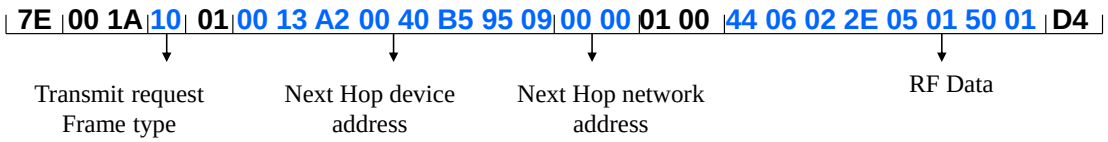



Figure 3.20: Router appending the Received data with its path

The appended frame is as shown in figure 3.20, where in RF data '50 01' indicates 'P' in path to coordinator node R1 is present. As the coordinator receives these data frames, it extracts the information of which sensor data is from which node and which path did this data take to reach it.

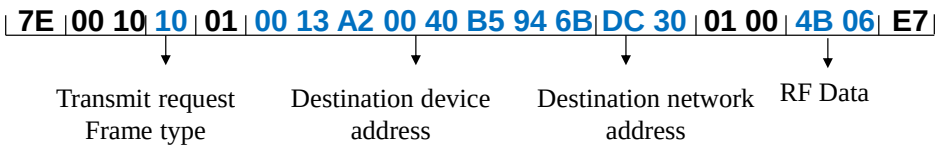



Figure 3.21: Transmitting the acknowledgment frame

And an acknowledgment is sent back to all the nodes from which these frames were received. The Acknowledgment frame is as shown in figure 3.21. In the frame, the payload or the RF data consists of 'K' with hex value '4B', followed by the list of NI of the nodes in the path taken by the data. Here, data takes a path of 6-1-0, hence C0 send acknowledgment frame to R1 which further passes the frame to node R6, assuring R6 that its data has been received by the coordinator.

Summarizing, in the example network all the nodes perform node discovery and gets aware of their neighbor nodes. Then they perform node classification after having received a neighbor list from its parent nodes and then transmitting its neighbor list to child nodes. After node classification, if threshold crossed, sensor data is sent through a particular path to coordinator. Through the same path acknowledgment is received back.

### 3.3.1 Summary of frames used

- **Begin Frame** : 7E 00 0F 10 01 64-bit broadcast address 16-bit broadcast address 01 00 'B' B3

- **Broadcast Hop Frame** : 7E 00 05 08 01 'B' 'H' 01 6B

- **Node Discovery Frame** : 7E 00 04 08 01 'N' 'D' 64

- **Node Discovery Response Frame** : 7E 00 19 88 01 'N' 'D' 00 16-bit address 64-bit address NI FF FE 00 00 C1 05 10 1E checksum

- **Neighbor List Frame** : 7E length 10 01 64-bit destination address 16-bit destination address 01 00 'S' 'NI' 'Neighbor List' checksum

- **Remote ATIS Command Frame** : 7E 00 0F 17 01 64-bit destination address 16-bit destination address 02 'I' 'S' checksum

- **Remote ATIS Response Frame** : 7E 00 15 97 01 64-bit Sender's address 16-bit Sender's address 'I' 'S' 00 'No. of samples' 'Digital Channel Masks' 'Analog Channel Mask' 'Analog Channel Data' checksum

• **Data Frame** : 7E length 10 01 64-bit destination address 16-bit destination address 01 00 'D' 'NI' 'Sensor value' 'P' 'NI' 'D' 'NI' 'Sensor value' checksum

• **Acknowledgment Frame** : 7E length 10 01 64-bit destination address 16-bit destination address 01 00 'K' 'List of NIs in the path' checksum

## 3.4 Flowcharts

Figures 3.22 and 3.23 illustrate the sequential steps followed by the coordinator and router in the routing algorithm developed.

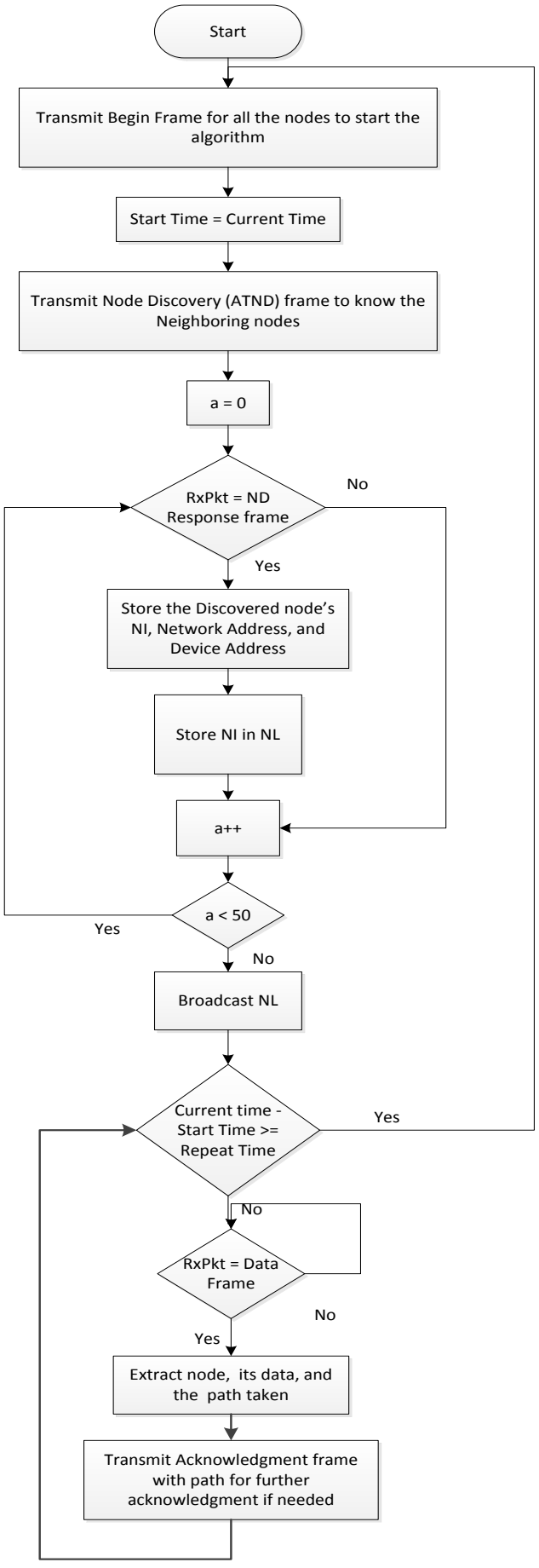
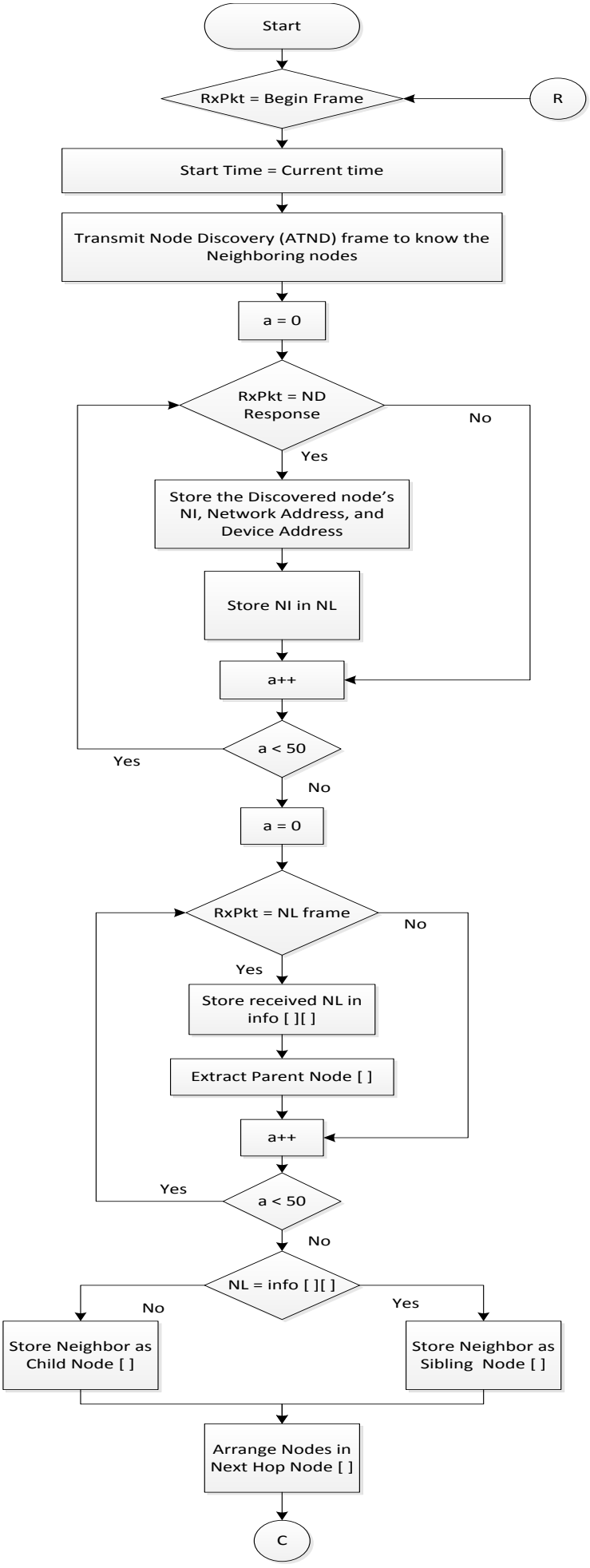



Figure 3.22: Coordinator Flowchart


Start
RxPkt = Begin Frame
R
Start Time = Current time
Transmit Node Discovery (ATND) frame to know the Neighboring nodes
a = 0
RxPkt = ND Response
No
Yes
Store the Discovered node's NI, Network Address, and Device Address
Store NI in NL
a++
a < 50
Yes
No
a = 0
RxPkt = NL frame
No
Yes
Store received NL in info [ ][ ]
Extract Parent Node [ ]
a++
Yes
a < 50
No
NL = info [ ][ ]
No
Yes
Store Neighbor as Child Node [ ]
Store Neighbor as Sibling Node [ ]
Arrange Nodes in Next Hop Node [ ]
C

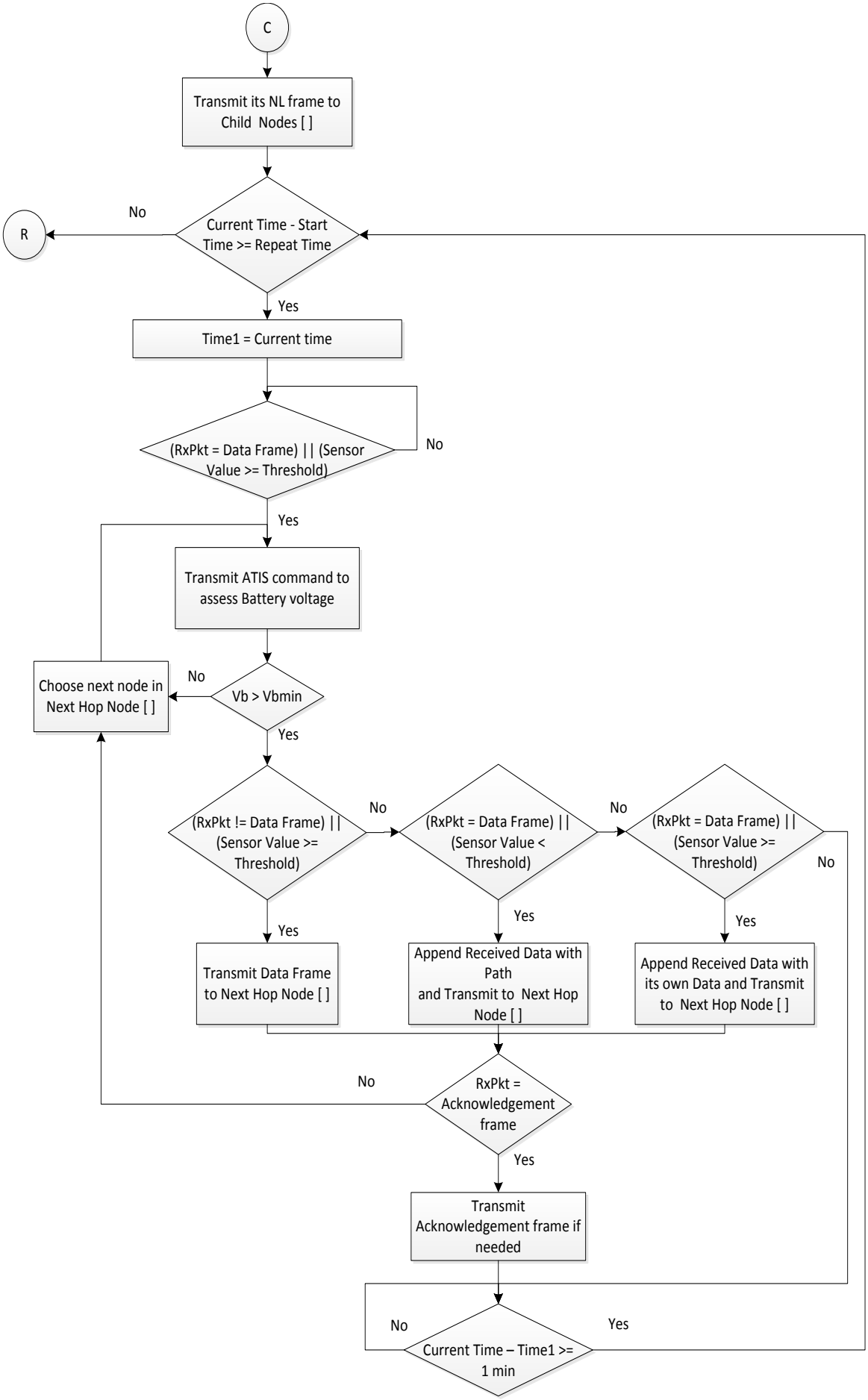



Figure 3.23: Router Flowchart

# CHAPTER 4

# TESTS AND RESULTS

## 4.1 Range Tests

The XBee range test gives an rough idea whether the communication link is strong or weak. Link Quality depends on multiple factors and is difficult to asses. Before deploying a network, performing a range test is extremely helpful as range test gives an insight of an existing weak link due to environmental scattering, distance limitations or interferences which can be rectified by installing additional router in between to fill in the gap.

Range Test measures Received Signal Strength Indicator (RSSI) uses following procedure to find it:

**1.** Transmit a Packet to the remote XBee

**2.** Receive the transmit status

**3.** Receive the sent packet back from the remote XBee

**4.** Get RSSI value using local ATDB command

Before testing the routing protocols, the following basic tests were performed to asses the link quality in the area of deployment.

RSSI variation with obstacles and distance are presented as follows:

### 4.1.1 RSSI Vs Distance

Two XBee modules were kept at fixed distance apart and RSSI test was performed using X-CTU software. Lesser the RSSI value in magnitude better is the link between the XBees. The RSSI levels at different distances have been illustrated in the graph4.1. RSSI was found to be reducing gradually with distance.

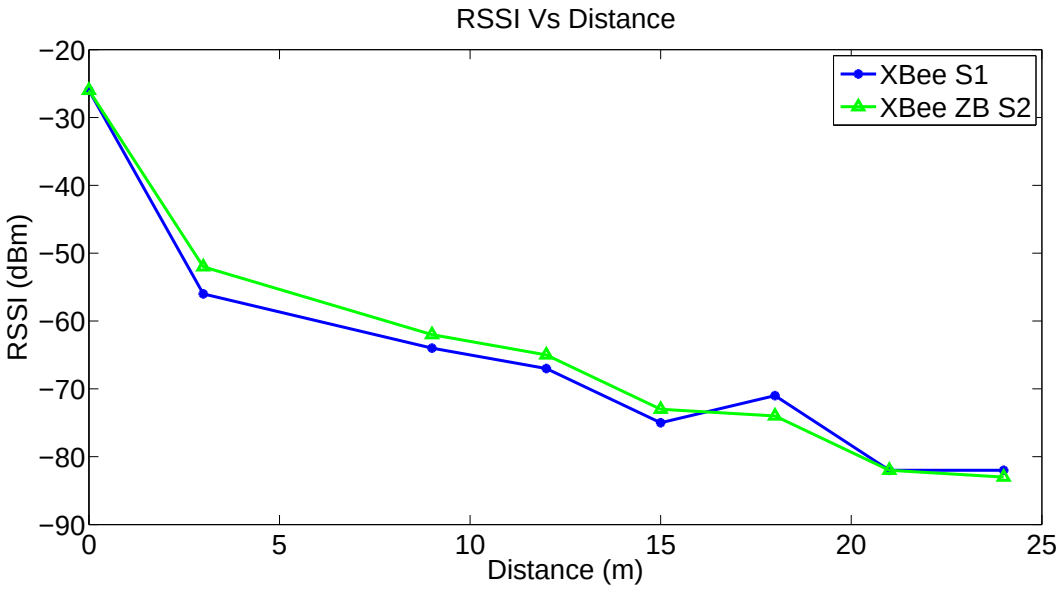



Figure 4.1: RSSI Variation with Distance (line of sight)

From the test, it was inferred that RSSI value between -23 dBm to -88 dBm is good enough and if RSSI level is beyond -88 dBm, the packets are dropped. Though the range for XBee S1 modules is specified to be 100 m and 120 m for XBee S2 ZB in line of sight condition, it is suggested to place the modules not farther than 90 m for ensured communication.

### 4.1.2 RSSI Vs Obstacles

Two XBee separated by obstacles were tested as shown in figure 4.2

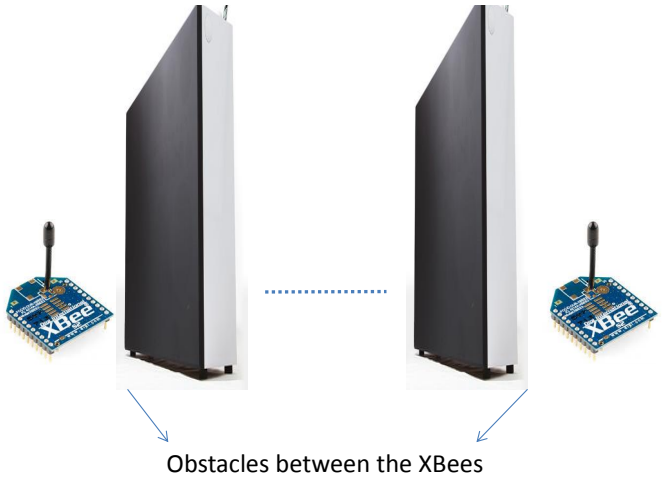



Figure 4.2: Test Setup for RSSI Vs Obstacles

RSSI value reached -92 dBm and packets were lost with around 6 walls in between,

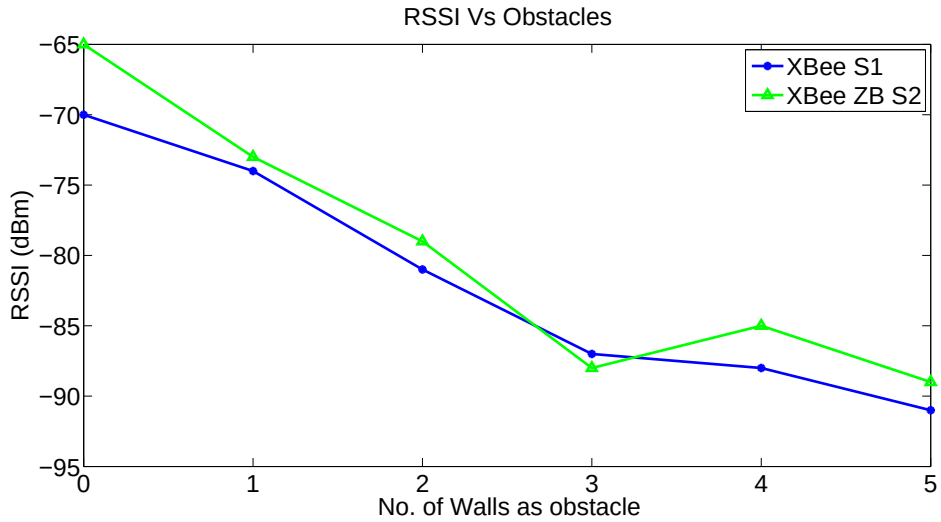



Figure 4.3: RSSI Variation with number of Obstacles

hence for indoor deployment, placing needs to be considered.

## 4.2 Roud Trip Time Tests

Round Trip Time(RTT) is the time taken for a packet to travel from a XBee to another and back to teh one transmitted the packet. This time can be calculated from Range Test frame list.

$$RTT = T_r - T_t \tag{4.1}$$

where, $T_r$ is the time-stamp at which XBee receives back the transmitted packet

$T_t$ is the time-stamp at which XBee transmits the packet

### 4.2.1 RTT Vs Payload Size

The packet size being transmitted by XBee, is one of the factors on which RTT depends. Using the range test, RTT is calculated for both XBee S1 and XBee ZB S2 and is plotted as shown in figure 4.5. The test setup is as shown in figure 4.4. The XBee modules were kept 4m apart.

It was observed that the RTT increases with the payload size and beyond payload size of 72 XBee raises 'payload too large' error. Hence, while programming the payload

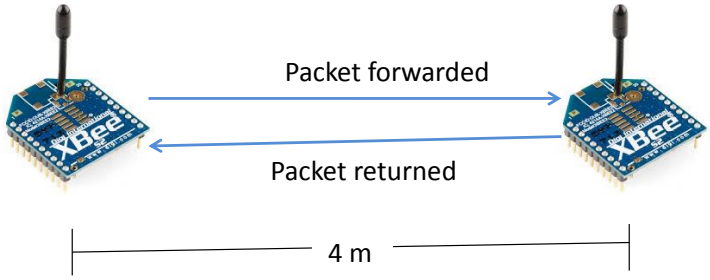



Figure 4.4: Test Setup for RTT Vs Payload variation

size should be limited accordingly.

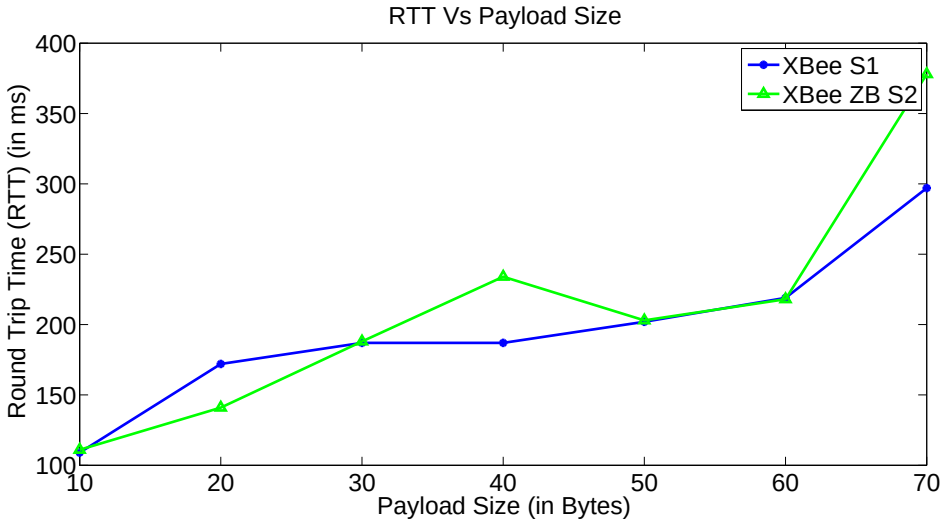



Figure 4.5: Round Trip Time variation with payload size

RTT was found to vary with payload size only, no significant variation with distance or hop or obstacle was observed.

## 4.3 Zigbee

A prototype testing with XBee S2 ZB modules was done. The modules were scattered in multiple ways assuring every node has a path to the coordinator either directly or through other nodes. Nodes were found to route each others data.

From the figure 4.6, it is clear that sensor data of only the nodes which cross the preset threshold is received, rest of the nodes do not transmit the data and hence not recorded in coordinators' serial monitor.

A field test at IIT Madras Stadium was also performed with 14 nodes and was found to work according to the need of MetroNet.

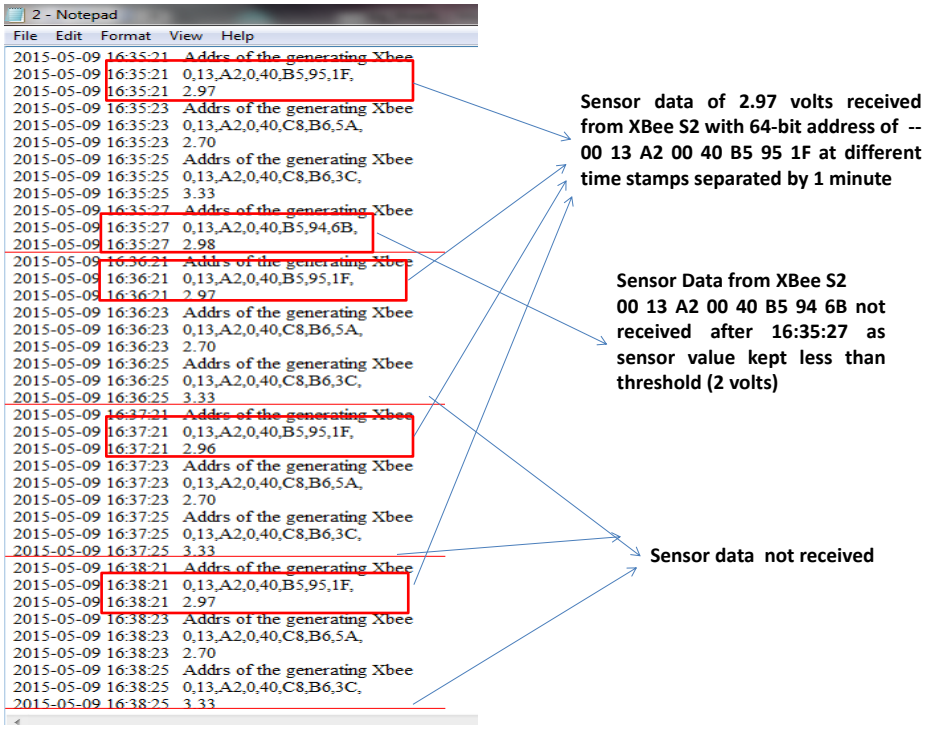



Figure 4.6: Sensor Data being Received by nodes with threshold crossed every 1 minute and Sensor data not received when sensor value less than threshold

**Power Consumption**

Current consumed was measured using a power meter and the total current consumed by the node while transmitting data was found to be 79.55 mA.

## 4.4 Digimesh

A prototype testing with XBee S1 digimesh modules was done. The modules were scattered in multiple ways assuring every node has a path to the coordinator either directly or through other nodes. The tests were centered to asses the following:

- Network in sleep sync or not.
- Whether sensor data is being transmitted as programmed.
- Nodes routing each others data when needed i.e, when node is not in direct communication range.
- Nodes transmitting the sensor data only if they cross their preset threshold.

All the tests conducted showed positive results for all the above assessed criteria. The entire network sleeps for 45 sec and was awake for the next 15 sec, as configured. Sensor data is to be transmitted every 1 min if the threshold is crossed and was accordingly programmed. Nodes were found to route each others data. The figure 4.7 shows the

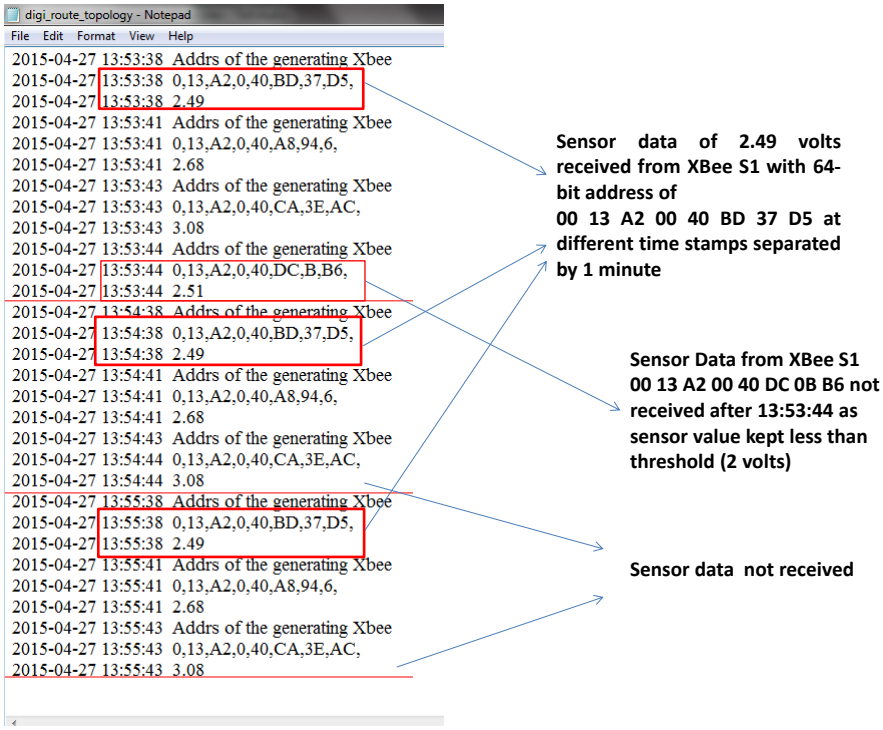



Figure 4.7: Sensor Data being Received by all nodes every 1 minute only on crossing threshold

sensor data and the corresponding XBee S1 address received by the sleep coordinator along with their time stamps. Each XBee sends sensor value which is above threshold every 1 minute.

### Power Consumption

The XBee S1 DigiMesh nodes were found to consume a current of 35.93 mA when in synchronous sleep and 90.15 mA when awake. Hence, increasing the battery life of the node.

## 4.5 Developed Routing Algorithm

Routing protocol was developed to learn the route taken by sensor data while reaching the coordinator or the data aggregator node. The path taken can be seen in coordinator

arduino's serial monitor screen-shot figure 4.8.

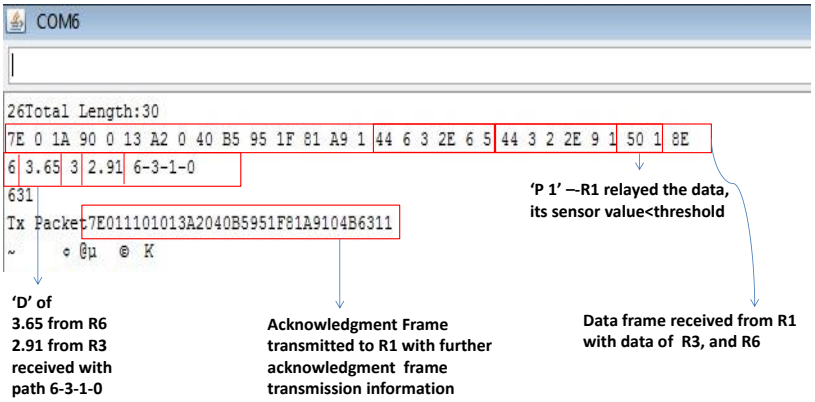



Figure 4.8: Path taken by the data detected at coordinator

Though the algorithm support path detection, it also sends sensor data only if threshold is crossed with a time interval of 1 minute. This can be viewed in figure 4.9.

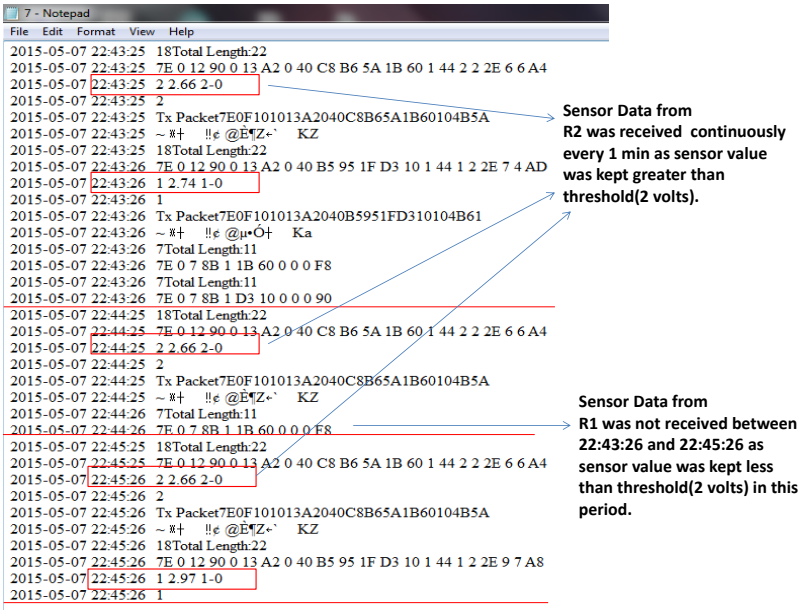



Figure 4.9: Data Received by coordinator

Current consumed by the nodes in this algorithm was found to be same as that in zigbee implementation, 79.55 mA. In addition to this, the voltage divider circuit used for assessing the battery voltage also consumes a considerable amount of current.

# CHAPTER 5

# CONCLUSION AND FUTURE WORK

## 5.1 Conclusion

Sensor data was transmitted at regular intervals only when it crosses a preset threshold. This was accomplished in all of the three implementations: ZigBee, DigiMesh and the routing algorithm developed. While testing the algorithms, node placement criteria was assessed on the basis of certain preliminary range and delay tests. ZigBee implementation reduces the power consumption of the WSN node when compared to the existing GSM. DigiMesh incorporates sleep mode operation and further reduces the power consumption.

Apart from sensing and reporting data at regular intervals when threshold is crossed, the routing algorithm developed, detects the route taken by sensor data while reaching the data aggregator. Also, the existing routing algorithms fails to detect node/ link failures which the developed routing can, through path detection. But, the routing algorithm developed consumes same power as Zigbee. However, it ensures that a node having a drained battery does not participate in routing the other nodes' data.

If power consumption aspect is considered, implementation with Digimesh sleep operation would be the best. And if node/ link failure detection is of prime importance for taking necessary measures to correct it, then use of the developed algorithm would give better results.

## 5.2 Future Scope

An integrated working of Digimesh sleep mode and the developed routing algorithm can be developed in future as it would give the best results in terms of both power consumption and node/ link failure detection.

Real Time Clock (RTC) interface with micro-controller can also be tried to provide time based accuracy. Replacing the voltage divider circuit for battery assessment with a low power consuming circuit will reduce the power consumption further. The existing radiation sensor module's integration with the developed short range communication node can be done.

# APPENDIX A

# ZIGBEE AND DIGIMESH

Listing A.1: Coordinator API

```
int adr;//address of generating XBee
void setup(){
Serial.begin(9600);
}

void loop(){
if (Serial.available()>=19){
if (Serial.read() ==0x7E){
for(int i=1;i<4;i++){
byte discardByte= Serial.read();
}
Serial.println("Address of the generating XBee");
for(int i=4;i<12;i++){//reading the address
adr= Serial.read();
Serial.print(adr,HEX);
Serial.print(",");
}
Serial.println();
for(int i=12;i<15;i++){//discard the unneccesary bytes
byte discardByte= Serial.read();
}
Serial.print((char)Serial.read());//sensor data being noted
Serial.print((char)Serial.read());
Serial.print((char)Serial.read());
Serial.println((char)Serial.read());
}
}
}
```

Listing A.2: Router AT

```
#include <avr/power.h>
#include <avr/sleep.h>
int sensr=0;//taking sensor value in analog input AD0
void setup()
{
Serial.begin(9600);
DDRD &= B00000011; // set Arduino pins 2 to 7 as inputs, leaves 0 & 1 (RX & TX) as is
DDRB = B00000000; // set pins 8 to 13 as inputs
PORTD |= B11111100; // enable pullups on pins 2 to 7
```

```
PORTB |= B11111111; // enable pullups on pins 8 to 1
set_sleep_mode(SLEEP_MODE_IDLE);// sleep mode is set here
sleep_enable();// enables the sleep bit in the mcucr register
power_spi_disable();
power_timer1_disable();
power_timer2_disable();
power_twi_disable();
sleep_mode(); // here the device is actually put to sleep
}

void loop()
{
int reading = analogRead(sensr);//sampling the sensor value
float voltage = reading *5;
voltage/=1024.0;//converting ADC value to actual value
if(voltage >=2)
{
Serial.print(voltage);//sending volatge value
delay(60000);  //every 1 min
}
set_sleep_mode(SLEEP_MODE_IDLE); // sleep mode is set here
sleep_enable(); // enables the sleep bit in the mcucr register
power_spi_disable();
power_timer1_disable();
power_timer2_disable();
power_twi_disable();
sleep_mode(); // here the device is actually put to sleep!!
}
```

# APPENDIX B

# ROUTING ALGORITHM DEVELOPED

Listing B.1: Coordinator API

```
byte nodeTable[8][12]; // 8 imply maximum possible no. of neighbors+1
byte RxPkt[54];
byte NL[8];// 8 imply maximum possible no. of neighbors+1
int RxPktLnth; //to store the total length(in bytes) of the packet including SD, chksum, length
int i=0,j,a,x;
int r;int c=0;
unsigned long start_time;
unsigned long intrvl_nt=300000;//interval for repeating the entire algorithm = 5 min

void setup(){
  Serial.begin(9600);
}

void loop(){
delay(2000);
TxBegin();//transmit begin frame
delay(200);
NodeDisc();
}

//TRANSMIT BEGIN FRAME
void TxBegin(){
byte NI=0;
byte chksm=0;
byte cheksum;
byte txdata[]={0x7E, 0x00, 0x00, 0x10, 0x01, 0,0,0,0,0,0,0xFF,0xFF,0xFF,0xFC,0,0,'B',0 };
txdata[2]=15;
for(i=3;i<18;i++)//checksum calculation
chksm+=txdata[i];
cheksum=0xFF-chksm;
txdata[i]=cheksum;
Serial.print("Tx Packet");
for(i=0;i<19;i++)
Serial.print(txdata[i],HEX);
Serial.println();
delay(200);
Serial.write(txdata,19);//transmitting begin frame
Serial.println();
delay(500);
}
```

```
//Receive all the Packets
void Rx(){
if(Serial.available()){
delay(50);
if(Serial.read()==0x7E){
RxPkt[0]=0x7E;
for(i=1;i<3;i++){
RxPkt[i]=Serial.read();
}
Serial.print(RxPkt[2]);
RxPktLnth=RxPkt[2]+4; // 4 for chksum-1,SD-1,length field-2 bytes
Serial.print("Total␣Length:");
Serial.println(RxPktLnth);
while(i<RxPktLnth){
RxPkt[i]=Serial.read();
i++;
}
for(i=0;i<RxPktLnth;i++){
Serial.print(RxPkt[i],HEX);
Serial.print("␣");
}
Serial.println();
}
}
delay(50);
}

//NODE DISCOVERY
void NodeDisc(){
start_time=millis();
int k,l,x=0;
int nd=0;a=0;
for(k=0;k<8;k++){
for(l=0;l<12;l++){
nodeTable[k][l]=0;
}
}
byte broadHops[] = { 0x7E, 0x00, 0x05, 0x08, 0x01, 0x42, 0x48, 0x01, 0x6B }; //ATBH1 (BH=1)
byte discTimeout[] = { 0x7E, 0x00, 0x05, 0x08, 0x01, 0x4E, 0x54, 0x50, 0x04 }; // ATNT50 (NT=8s)
byte applychanges[] = { 0x7E, 0x00, 0x04, 0x08, 0x01, 0x41, 0x43, 0x72 }; // ATAC
byte nodeDisc[] = { 0x7E, 0x00, 0x04, 0x08, 0x01, 0x4E, 0x44, 0x64 }; // ATND
Serial.write(broadHops,9);// Sending ATBH1 to local XBee
Serial.println();
delay(100);
Serial.write(discTimeout,9);// Sending ATNT50 to local XBee
Serial.println();
delay(100);
Serial.write(applychanges,8);
Serial.println();
delay(100);
```

```
Serial.write(nodeDisc,8);//Sending ATND to local XBee
Serial.println();
delay(100);
while(a<50 ){//receive all the Neighbor Node Information
r=0;
for(i=0;i<54;i++)
RxPkt[i]=0;
delay(200);
while(r!=1 && Serial.available()){// waiting for ATND response frame
Rx();
if((RxPkt[5]=='N') && (RxPkt[6]=='D'))
r=1;
}
if((RxPkt[5]=='N') && (RxPkt[6]=='D')){
for(l=0;l<2;l++){
nodeTable[nd][l]=RxPkt[l+18];//Storing NI
}
for(l=0;l<2;l++){
nodeTable[nd][l+2]=RxPkt[l+8];//Storing 16 bit ntwrk addrs
}
for(l=0;l<8;l++){
nodeTable[nd][l+4]=RxPkt[l+10];//Storing 64 bit addrs
}
nd++;
}
a++;
}
if(nd>1){// more than one node detected then store Neighbor list
Serial.println("Node␣TAble:");
for(k=0;k<nd;k++){
if(nodeTable[k][0]==82)
Serial.print("R");
if(nodeTable[k][0]==67)
Serial.print("C");
if(nodeTable[k][1]>=48 && nodeTable[k][1]<58){//storing Neighbor List
Serial.print((nodeTable[k][1])-48);
NL[k]=((nodeTable[k][1])-48);
c++;
}
else{
Serial.print((nodeTable[k][1]));
NL[k]=nodeTable[k][1];
c++;
}
Serial.print("␣");
for(l=2;l<12;l++){//printing Node Table
Serial.print(nodeTable[k][l],HEX);
Serial.print("␣");
}
Serial.println();
}
```

```
Serial.print("Neighbor List:");
for(i=1;i<c;i++){//printing Neighbor List
Serial.print(NL[i]);
Serial.print(" ");
}
delay(800);
TxNL(c-1);
}
}

//TRANSMITTING THE NL FRAME
void TxNL(int length){
byte NI=0;
byte chksm=0;
byte cheksum;
byte txdata[]={0x7E, 0x00, 0x00, 0x10, 0x01, 0,0,0,0,0,0,0xFF,0xFF,0xFF,0xFC,1,0,'S',0,0 };
txdata[2]=length+16;
txdata[18]=NI;//placing NI in frame
for(i=0;i<length;i++)//putting neighbor list into frame
txdata[i+19]=NL[i+1];
for(i=3;i<length+19;i++)//checksum calculation
chksm+=txdata[i];
cheksum=0xFF-chksm;
txdata[i]=cheksum;
Serial.print("Tx Packet");
for(i=0;i<length+20;i++)
Serial.print(txdata[i],HEX);
Serial.println();
delay(200);
Serial.write(txdata,(length+20));//transmiting the NL frame
Serial.println();
delay(500);
r=0;
while(r!=1){//waiting for data frames
if(Serial.available()){
Rx();
if((RxPkt[3]==0x90) && (RxPkt[15]=='D')){
r=1;
DataPath();
return;
}
}
}
}

//EXTRACT DATA AND PATH FROM THE PACKET
void DataPath(){
int length,i,k,l,a,r;
byte NI=0;
byte path[]={0,0,0,0,};
a=0;
```

```
while(a<150 && ((unsigned long)millis()-start_time)<=intrvl_nt){//continue till 5min
length=RxPkt[2]-12;
//generating node and corresponding data
for(i=0;i<length;i++){
if(RxPkt[i+15]=='D'){
Serial.print(RxPkt[i+16]); //node ID
Serial.print("␣");
for(k=i+17;k<i+17+4;k++){
if(k==i+18)
Serial.print(char(RxPkt[k])); //data
else
Serial.print(RxPkt[k]);
}
Serial.print("␣");
}
}
//extracting path information from the frame
k=0;
for(l=0;l<length;l++){
if(RxPkt[l+15]=='D'|| RxPkt[l+15]=='P'){
path[k]=RxPkt[l+16];
k++;
Serial.print(RxPkt[l+16]);
Serial.print("-");
}
}
Serial.print(NI);
Serial.println();
for(i=0;i<k;i++)
Serial.print(path[i]);
Serial.println();
TxAck(path,k);//send acknowledgment frame
for(i=0;i<RxPktLnth;i++)
RxPkt[i]=0;
r=0;
//continue waiting for data frames till 5min time has not expired
while(r!=1 && ((unsigned long)millis()-start_time)<=intrvl_nt){
if(Serial.available()){
Rx();
if((RxPkt[3]==0x90) && (RxPkt[15]=='D')){
r=1;
break;
}
}
}
a++;
}
}

//TRANSMIT ACKOWLEDGMENT FRAME
void TxAck(byte path[],int k){
```

```
byte NI=0;
int i,l;
byte chksm=0;
byte cheksum;
byte txack[]={0x7E, 0x00, 0x00, 0x10, 0x01, 0,0,0,0,0,0,0,0,0,1,0,'K',0,0 };
txack[2]=k+14;
for(i=0;i<(k-1);i++)
txack[i+18]=path[i];//place rest of the path in the frame for forwarding ack frame
for(l=0;l<c;l++){
if(path[i]==nodeTable[1][1]-48)
break;
}
for(j=0;j<8;j++)//set destination address with node from where data frame has been received
txack[j+5]=RxPkt[j+4];
for(j=0;j<2;j++)
txack[j+13]=RxPkt[j+12];
for(i=3;i<k+17;i++)//checksum calculation
chksm+=txack[i];
cheksum=0xFF-chksm;
txack[i]=cheksum;
Serial.print("Tx Packet");
for(i=0;i<k+18;i++)
Serial.print(txack[i],HEX);
Serial.println();
Serial.write(txack,(k+18));//transmitting ack frame
Serial.println();
}
```

Listing B.2: Router API

```
int sensr=0;// sensor output connected to analog pin AD0
byte nodeTable[8][12];// 8 imply maximum possible no. of neighbors+1
byte NextHop[]={0,0,0,0,0,0,0,0};
byte RxPkt[54];
byte NL[8];// 8 imply maximum possible no. of neighbors+1
byte childNode[]={0,0,0,0,0,0,0,0};
int RxPktLnth; //to store the total length(in bytes) of the packet including SD, chksum, length
int i=0;
int r,c,s,x,j,l;
byte NI;//node identification
unsigned long txtime1;
unsigned long txtime2;
unsigned long intrvl_data=60000;// interval for sending data = 1 min
unsigned long intrvl_nt=300000;//interval for repeating the entire algorithm = 5 min
unsigned long start_time;

void setup(){
Serial.begin(9600);
Serial.println("Waiting for Begin Frame");
}
```

```
void loop(){
delay(200);
r=0;
while(r!=1){// waiting for the begin frame to arrive
if(Serial.available()){
Rx();
if((RxPkt[15]=='B') && (RxPkt[3]==0x90))
r=1;
}
}
}

//Receive all the Packets
void Rx(){
if(Serial.available()){
delay(50);
if(Serial.read()==0x7E){
RxPkt[0]=0x7E;//Start Delimiter of the frame
for(i=1;i<3;i++){//length field of the lacket
RxPkt[i]=Serial.read();
}
Serial.print(RxPkt[2]);
RxPktLnth=RxPkt[2]+4; // 4 for chksum-1,SD-1,length field-2 bytes
Serial.print("Total␣Length:");
Serial.println(RxPktLnth);
while(i<RxPktLnth){ //storing the full frame into RxPkt [] array
RxPkt[i]=Serial.read();
i++;
}
for(i=0;i<RxPktLnth;i++){ //printing the frame
Serial.print(RxPkt[i],HEX);
Serial.print("␣");
}
Serial.println();
if((RxPkt[15]=='B') && (RxPkt[3]==0x90)) //on receving the begin frame start Node Discovery
NodeDisc();
}
}
delay(50);
}

// NODE DISCOVERY PROCESS
void NodeDisc(){
start_time=millis(); //record the time at which node discovery process started
int k,l,c=0;
int nd=0,a=0;
for(k=0;k<8;k++){//clear node table
for(l=0;l<12;l++){
nodeTable[k][l]=0;
}
```

```
}
byte broadHops[] = {0x7E,0x00,0x05,0x08,0x01,0x42,0x48,0x01,0x6B}; //ATBH1 (BH=1)
byte discTimeout[] = {0x7E,0x00,0x05,0x08,0x01,0x4E,0x54,0x50,0x04}; // ATNT50 (NT=8s)
byte nodeDisc[] = {0x7E,0x00,0x04,0x08,0x01,0x4E,0x44,0x64}; // ATND
Serial.write(broadHops,9); // Sending ATBH1 to local XBee
Serial.println();
delay(200);
Serial.write(discTimeout,9);// Sending ATNT50 to local XBee
Serial.println();
delay(200);
Serial.write(nodeDisc,8);//Sending ATND to local XBee
Serial.println();
delay(200);
while(a<50){ //receive all the Neighbor Node Information
r=0;
if((RxPkt[5]=='N') && (RxPkt[6]=='D')){
for(i=0;i<54;i++)
RxPkt[i]=0;
}
delay(200);
while(r!=1 && Serial.available()){// waiting for ATND response frame
Rx();
if((RxPkt[5]=='N') && (RxPkt[6]=='D'))
r=1;
}
if((RxPkt[5]=='N') && (RxPkt[6]=='D')){
for(l=0;l<2;l++){
nodeTable[nd][l]=RxPkt[l+18];  // Storing NI
}
for(l=0;l<2;l++){
nodeTable[nd][l+2]=RxPkt[l+8];  //Storing 16 bit ntwrk address
}
for(l=0;l<8;l++){
nodeTable[nd][l+4]=RxPkt[l+10];   //Storing 64 bit address
}
nd++;
}
a++;
}
c=0;
if(nd>1){// more than one node detected then store Neighbor list
Serial.println("Node␣TAble:");
for(k=0;k<nd;k++){
if(nodeTable[k][0]==82)
Serial.print("R");
if(nodeTable[k][0]==67)
Serial.print("C");
if(nodeTable[k][1]>=48 && nodeTable[k][1]<58){//storing Neighbor List
Serial.print((nodeTable[k][1])-48);
NL[k]=((nodeTable[k][1])-48);
c++;
```

```
}
else{
Serial.print((nodeTable[k][1]));
NL[k]=nodeTable[k][1];
c++;
}
Serial.print("␣");
for(l=2;l<12;l++){//printing Node Table
Serial.print(nodeTable[k][l],HEX);
Serial.print("␣");
}
Serial.println();
}
NI=NL[0]; //its own Node ID
Serial.print("Neighbor␣List:");
for(i=1;i<c;i++){//printing Neighbor List
Serial.print(NL[i]);
Serial.print("␣");
}
r=0;
while(r!=1){//waiting for Neighbor List frame
if(Serial.available()){
Rx();
if((RxPkt[3]==0x90) && (RxPkt[15]=='S')){
r=1;
FindNextHop(c,RxPkt);
return;
}
}
}
}
}

//NODE CLASSIFICATION PROCESS
void FindNextHop(int c, byte RxPkt[]){
i=0;
x=0;
r=0;
l=0;
int a;
int j,k,y;
int b=0;
int len=0;
byte info[4][6]={{'B','B','B','B','B','B'},{'B','B','B','B','B','B'},
                 {'B','B','B','B','B','B'},{'B','B','B','B','B','B'}};
byte parentNode[]={0,0,0,0,0,0,0};
byte Sib[]={0,0,0,0,0,0,0};
int p=0,h=0,cn=0,w;
Serial.print("no.␣of␣neighbr␣nodes");
Serial.println(c-1);
j=0;
```

```
while(x<8){
if((RxPkt[3]==0x90) && (RxPkt[15]=='S')){
for(l=0;l<(RxPkt[2]-13);l++){//storing the neighbor list of the source and source information
info[j][l]=RxPkt[l+16];
}
j++;
}
x++;
delay(500);
for(i=0;i<54;i++)
RxPkt[i]=0;
r=0;
while(r!=1 && Serial.available()){//waiting for Neighbor List frame
Rx();
if((RxPkt[3]==0x90) && (RxPkt[15]=='S')){
r=1;
break;
}
}
}
h=0;cn=0;
for(y=1;y<c;y++){//finding parent, sibling and child nodes
x=0;
for(j=0;j<4;j++){
for(l=0;(l<6 && info[j][1]!='B');l++){
if(NL[y]==info[j][l]){
x=1;
break;
}
}
if(x==1)
break;
}
if(x==1){
Sib[h]=NL[y];
h++;
}
else{
childNode[cn]=NL[y];
cn++;
}
}
p=0;
for(j=0;(j<4 && info[j][0]!='B');j++){
parentNode[p]=info[j][0];
p++;
}
i=0;//finding Next hop nodes
while(i<p){
NextHop[i]=parentNode[i];
i++;
```

```
}
for(l=0;l<h;l++){
x=0;
for(r=0;r<p;r++){
if(parentNode[r]==Sib[l])
x=1;
}
if(x!=1){
NextHop[i]=Sib[l];
i++;
}
}
l=0;
while(l<cn){
NextHop[i]=childNode[l];
i++;
l++;
}
len=i;
Serial.print("Parent␣Nodes:");
for(i=0;i<p;i++){
Serial.print(parentNode[i]);
Serial.print("␣");
}
Serial.println();
Serial.print("Child␣Nodes:");
for(i=0;i<cn;i++){
Serial.print(childNode[i]);
Serial.print("␣");
}
Serial.println();
Serial.print("Siblings:");
for(i=0;i<h;i++){
Serial.print(Sib[i]);
Serial.print("␣");
}
Serial.println();
Serial.print("Next␣Hop␣Nodes:");
for(i=0;i<len;i++){
Serial.print(NextHop[i]);
Serial.print("␣");
}
Serial.println();
delay(500);
if(cn>0){// NL frame transmission to all the child nodes
for(w=0;w<cn;w++){
w=TxNLtoChild(w,cn,c);
delay(100);
}
}
do{
```

```
txtime1=millis();//record time at which 1st data transmission is taking place
DataTx(NextHop,len,c);
r=0;a=0;s=0;
while(s<50){
while(r!=1 && a<150){// wait for data frame
if(Serial.available()){
Rx();
if((RxPkt[3]==0x90) && (RxPkt[15]=='D'))
r=1;
}
a++;
}
if((RxPkt[3]==0x90) && (RxPkt[15]=='D'))
DataTx(NextHop,len,c);
s++;
}
txtime2=millis();
while(1){//repeat data transmission every 1min
if((((unsigned long)millis()-txtime1)==intrvl_data)||((unsigned long)millis()-start_time)>=intrvl_nt)
break;
if(Serial.available()){//waiting for data frame
Rx();
if((RxPkt[3]==0x90) && (RxPkt[15]=='D'))
DataTx(NextHop,len,c);
}
}
}while(((unsigned long)millis()-start_time)<=intrvl_nt);//repeat only till 5min time expires
}

//NL FRAME TRANSMISSION TO CHILD NODES
int TxNLtoChild(int w,int cn,int length){
int k=0;
byte chksm=0;
byte cheksum;
byte txdata[]={0x7E, 0x00, 0x00, 0x10, 0x01, 0,0,0,0,0,0,0,0,0,1,0,'S',0,0,0,0,0,0,0,0,0};
txdata[2]=length+15;
txdata[18]=NI;
for(l=0;l<length;l++){//getting child nodes addresses
if(childNode[w]==nodeTable[l][1]-48)
break;
}
for(k=0;k<8;k++)//64 bit address
txdata[k+5]=nodeTable[l][k+4];
for(k=0;k<2;k++)//16 bit address
txdata[k+13]=nodeTable[l][k+2];
for(k=0;k<(length-1);k++)//adding Neighbor list to the frame
txdata[k+19]=NL[k+1];
for(k=3;k<length+18;k++)//checksum calculation
chksm+=txdata[k];
cheksum=0xFF-chksm;
txdata[k]=cheksum;
```

```
Serial.print("Tx␣Packet");
for(k=0;k<length+19;k++)
Serial.print(txdata[k],HEX);
Serial.println();
delay(100);
Serial.write(txdata,(length+19));//transmitting frame
Serial.println();
return w;
}

//DATA FRAME TRANSMISSION
void DataTx(byte NextHop[],int le,int c){
byte chksm=0;
byte cheksum,ch,chk;
int length;
long int Vb;
int y;
int source;
int t,k,a,g;
byte txdata[]={0x7E, 0x00, 0x00, 0x10, 0x01, 0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
Serial.println("Start");
for(i=0;i<le;i++){
Serial.print(NextHop[i]);
Serial.print("␣");
}
Serial.println();
int reading = analogRead(sensr);//ADC reading of sensor value
float voltage = reading *5;
voltage/=1024.0;//getting actual sensor value
while(r!=1 && Serial.available()){//waiting for data frame
Rx();
if((RxPkt[3]==0x90) && (RxPkt[15]=='D'))
r=1;
}
if(voltage>=2 || RxPkt[15]=='D'){
if(voltage<2 && RxPkt[15]=='D'){//data received but threshold not crossed
length=RxPkt[2]-12+2;//+2 for path bytes 'P NI'
if(RxPkt[length+7]=='D')
source=RxPkt[length+8];
if(RxPkt[length+11]=='P')
source=RxPkt[length+12];//knowing the source of data frame received
txdata[2]=length+14;///frame length of the packet
for(i=0;i<(length-2);i++)//add received data to the frame
txdata[i+17]=RxPkt[i+15];
txdata[i+17]='P';//adding a stamp of path
i++;
txdata[i+17]=NI;
}
if(voltage>=2 && RxPkt[15]=='D'){//data received and threshold crossed
length=RxPkt[2]-12+6;//+6 for 'D NI Data'
if(RxPkt[length+3]=='D')
```

```
source=RxPkt[length+4];
if(RxPkt[length+7]=='P')
source=RxPkt[length+8];//knowing the source of data
txdata[2]=length+14;
Serial.print(voltage);
a=voltage*100;
i=3;
while(i>=0){//voltage conversion into frame
if(i==1){
txdata[i+13+length]='.'; // . has hex value of 2E
i--;
}
else{
t=a%10;
txdata[i+13+length]=t;
a=a/10;
i--;
}
}
for(i=0;i<(length-6);i++)
txdata[i+17]=RxPkt[i+15];
txdata[i+17]='D';//identifying that its a data frame
i++;
txdata[i+17]=NI;//implying the generator of the data
Serial.println();
}
if(voltage>=2 && RxPkt[15]!='D'){//Threshold crossed but not received any data frame
length=6;
txdata[2]=length+14;
Serial.print(voltage);
a=voltage*100;
i=3;
while(i>=0){//voltage being put in frame
if(i==1){
txdata[i+19]='.'; // . has hex value of 2E
i--;
}
else{
t=a%10;
txdata[i+19]=t;
a=a/10;
i--;
}
}
Serial.println();
for(i=0;i<4;i++){
Serial.print(txdata[i+19]);
Serial.print("␣");
}
Serial.println();
txdata[17]='D';//identifying that its a data frame
```

```
txdata[18]=NI;//implying the generator of the data
source='B';/ias there is no source of data frame assigning it to'B'
}
Serial.println(source);
g=0;
while(g<le){
if(NextHop[g]!=source){//if next hop is not as the node sending data frame
for(l=1;l<c;l++){
if(NextHop[g]==nodeTable[l][1]-48)//comparing node table an dnext hop node
break;
}
Serial.println(NI);
Serial.println(NextHop[g]);
for(k=0;k<8;k++)//putting destination address into the frame
txdata[k+5]=nodeTable[l][k+4];
for(k=0;k<2;k++)
txdata[k+13]=nodeTable[l][k+2];
//BATTERY VOLTAGE ASSESSMENT
byte Vbattery[]={0x7E,0x00,0x0F,0x17,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x02,
                 'I','S',0x00};//remote ATIS command
for(k=0;k<8;k++)//putting address of next hop i.e, destination address into the frame
Vbattery[k+5]=nodeTable[l][k+4];
for(k=0;k<2;k++)
Vbattery[k+13]=nodeTable[l][k+2];
ch=0;chk=0;
for(k=3;k<18;k++)//cheksum calculation
ch+=Vbattery[k];
chk=0xFF-ch;
Vbattery[k]=chk;
Serial.print("Vbattery␣Packet");
for(k=0;k<19;k++){
Serial.print(Vbattery[k],HEX);
Serial.print("␣");
}
Serial.write(Vbattery,19);//trensmitting ATIS command
Serial.println();
y=0;
while(y!=1){//waiting for ATIS response frame
if(Serial.available()){
Rx();
if(RxPkt[3]==0x97 && RxPkt[15]=='I' && RxPkt[16]=='S'){
Vb=((RxPkt[RxPkt[2]+3-2]<<8) | RxPkt[RxPkt[2]+3-1]);
Vb=(Vb*1200)/1024;// Vb calculation
Serial.println(Vb);
y=1;;
}
}
}
if(Vb>500){//condition on minimum Vb required
chksm=0;cheksum=0;
for(k=3;k<length+17;k++)//cheksum calculation
```

```
chksm+=txdata[k];
cheksum=0xFF-chksm;
txdata[k]=cheksum;
Serial.print("Tx Packet");
for(k=0;k<length+18;k++){
Serial.print(txdata[k],HEX);
Serial.print(" ");
}
Serial.println();
delay(200);
Serial.write(txdata,(length+18));//transmitting the data frame
Serial.println();
delay(1000);
r=0;
x=0;
while(r!=1 && x<150){//waiting for acknowledgement frame
if(Serial.available()){
Rx();
if((RxPkt[3]==0x90) && (RxPkt[15]=='K')){
r=1;
Serial.println("Successfully transmited to Coordinator");
}
}
x++;
}
}
if((RxPkt[3]==0x90) && (RxPkt[15]=='K'))
break;
else
g++;//go to next node in next hop node array
}
}
if((RxPkt[3]==0x90) && (RxPkt[15]=='K')){
k=RxPkt[2]-13;
if(k>0)//if ack is to be forwarded to other nodes in path
TxAck(c);
}
}
}

//FORWARDING ACKOWLEDGEMENT TO OTHER NODES IN PATH
void TxAck(int c1){
int i,l,k,j;
byte chksm=0;
byte cheksum;
byte txack[]={0x7E, 0x00, 0x00, 0x10, 0x01, 0,0,0,0,0,0,0,0,0,1,1,'K',0,0 };
k=RxPkt[2]-13;
if(k>0){
txack[2]=15+k-1;
for(i=0;i<(k-1);i++)//informing about further path if present
txack[i+18]=RxPkt[i+16];
```

```
Serial.println(RxPkt[i+16]);
for(l=0;l<cl;l++){//finding address of the node to which ack is to be forwarded
if(RxPkt[i+16]==nodeTable[l][1]-48)
break;
}
for(j=0;j<8;j++)//putting destination address in frame
txack[j+5]=nodeTable[l][j+4];
for(j=0;j<2;j++)
txack[j+13]=nodeTable[l][j+2];
for(i=3;i<(k-1)+18;i++)//checksum calculation
chksm+=txack[i];
cheksum=0xFF-chksm;
txack[i]=cheksum;
Serial.print("Tx␣Packet");
for(i=0;i<(k-1)+19;i++)
Serial.print(txack[i],HEX);
Serial.println();
delay(200);
Serial.write(txack,((k-1)+19));//transmitting the frame
Serial.println();
}
}
```

# REFERENCES


[1] M. D. Patel, M. P. Ratheesh, M. S. Prakasha, S. S. Salunkhe, A. V. Kumar, V. D. Puranik, and C. Nair, "Multi-detector environmental radiation monitor with multichannel data communication for indian environmental radiation monitoring network (iermon)," 2011. [Online]. Available: http://www.barc.gov.in/publications/nl/2011/2011050606.pdf

[2] J. A. Gutierrez, "Ieee std. 802.15.4," 2005. [Online]. Available: http://www.cs.berkeley.edu/~prabal/teaching/cs294-11-f05/slides/day21.pdf

[3] S. Farahani, *Zigbee Wireless Networks and Transceivers*. Newnes Publications, 2008.

[4] D. A. Ott, "Wireless networking with ieee 802.15.4 and 6lowpan," 2012. [Online]. Available: http://www.cs.berkeley.edu/~prabal/teaching/cs294-11-f05/slides/day21.pdf

[5] Jennic, "Co-existence of ieee 802.15.4 at 2.4 ghz," 2008. [Online]. Available: http://www.jennic.com/files/support_files/JN-AN-1079%20Coexistence%20of%20IEEE%20802.15.4%20In%20The%202.4GHz%20Band-1v0.pdf

[6] *XBee/XBee-PRO Digimesh 2.4GHz RF Modules-Digi International*.

[7] C. Perkins and E. Belding-Royer, "Ad hoc on-demand distance vector (aodv) routing," 2003. [Online]. Available: https://www.ietf.org/rfc/rfc3561.txt

[8] E. E. Exploria, "Routing protocols in manets," 2013. [Online]. Available: http://www.eexploria.com/routing-protocols-in-manets/

[9] *XBee/XBee-PRO ZB RF Modules-Digi International*.

[10] tymkrs, "Xbee adapter board - pin connections," 2012. [Online]. Available: http://tymkrs.tumblr.com/post/17661586353/xbee-adapter-board-pin-connections