A LOW-COST WIRELESS SENSOR NETWORK SYSTEM USING RASPBERRY PI AND ARDUINO FOR ENVIRONMENTAL MONITORING APPLICATIONS

Sheikh Mohammad Ferdoush

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

May 2014

APPROVED:

Xinrong Li, Major Professor
Hyoung Soo Kim, Committee Member
Shengli Fu, Committee Member and Interim Chair of the Department of Electrical Engineering
Costas Tsatsoulis, Dean of the College of Engineering
Mark Wardell, Dean of the Toulouse Graduate School

Ferdoush, Sheikh Mohammad. *A Low-Cost Wireless Sensor Network System Using Raspberry Pi and Arduino for Environmental Monitoring Applications.* Master of Science (Electrical Engineering) , May 2014, 82 pp., 15 tables, 49 figures, references, 40 titles.

Sensors are used to convert physical quantity into numerical data. Various types of sensors can be coupled together to make a single node. A distributed array of these nodes can be deployed to collect environmental data by using appropriate sensors. Application of low powered short range radio transceivers as a communication medium between spatially distributed sensor nodes is known as wireless sensor network. In this thesis I build such a network by using Arduino, Raspberry Pi and XBee. My goal was to accomplish a prototype system so that the collected data can be stored and managed both from local and remote locations. The system was targeted for both indoor and outdoor environment.

As a part of the development a controlling application was developed to manage the sensor nodes, wireless transmission, to collect and store data using a database management service. Raspberry Pi was used as base station and webserver. Few web based application was developed for configuring the network, real time monitoring, and database management. Whole system functions as a single entity.

The use of open source hardware and software made it possible to keep the cost of the system low. The successful development of the system can be considered as a prototype which needs to be expanded for large scale environmental monitoring applications.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to the Almighty God for giving me the strength and keeping me agile to do this work.

I would like to express immense gratitude to my supervisor Dr. Xinrong Li. His strong support throughout my academic career at UNT made it possible to complete the thesis work in time. I would like to take this opportunity to thank him for his thoughtful suggestions, constant support, advice and patience.

I want to thank my committee members Dr. Shengli Fu, Dr. Hyoung Soo Kim and Dr. Yan Wan for their suggestions and help which played a vital role towards the completion of the thesis.

I am also grateful to all the faculty members and staffs of the Department of Electrical Engineering, University of North Texas who directly or indirectly contributed towards the completion of the thesis.

I would like to thank all my friends and fellow students at the department who helped me with their observation and inspired me during the time of despair.

I am especially grateful to my parents, wife and rest of my family members for their encouragement and support. Without their inspiration it would have been difficult for me to achieve my goal.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

The job of a sensor is to convert a physical quantity into numerical data. A single sensor or a number of different types of sensor can be integrated into one single device to collect data from the same spot which can be referred as a sensor node. When many sensor nodes are organized into a distributed network to collect data from a large indoor or outdoor environment we call this a sensor network. A data communication link through wire can be established for each of these nodes to transmit collected data to a central data collection sink node. The use of wire has become primitive and proved to be cumbersome for the sensor nodes residing far from the user.

In recent years, we have seen a new era of short range wireless technologies like Wi-Fi, Bluetooth, ZigBee [6], 6LoWPan [7] emerging in front of us. Some of them in fact have become few years older and quite familiar to us. The wireless devices made using these technologies have different data rate, range, power consumption and features. One thing is common though; they are small and can be easily attached to a sensor node.

When we use a bunch of sensor nodes with radio transceiver to collect and transmit physical or environmental data through wireless communication we call it wireless sensor network. Some of the uses of wireless sensor network are air/water pollution monitoring, building management system [8], environmental monitoring, etc.

The use of wireless sensor network introduces a few challenging issues. Sensor nodes are going to send data autonomously or when they are asked to do so. A methodology has to

be developed to define their characteristics. Data acquisition and data warehousing techniques are required to deal with the incoming data. A framework is required for accessing or presenting the collected data. In this thesis, I describe a system that has been developed to deal with all these issues.

I proposed and developed a web-based framework for control, acquisition and presentation of sensors and sensor data. The concept of web-based monitoring for wireless sensor network is nothing new. A web-based testbed [9] was created by Harvard researchers for wireless sensor network which handles reprogramming of sensor nodes, data logging and scheduling jobs on the testbed. In [10] Delicato et al. proposed a web services approach for the design of sensor networks, in which sensor nodes are service providers and applications are clients of such services. Software architecture was developed and a web-based application was designed to query a wireless sensor network in [11]. Authors of this paper have introduced two protocols; one for continuous monitoring and the other for query based monitoring. In [12] an IP-based wireless sensor network for web interfacing was developed. The use of representational state transfer web services and JavaScript Object Notation format is also shown in [13]. In this thesis I followed similar approaches but the new system is much effective in terms of cost and portability.

1.2 Objective of Research

The main objective of this thesis was to make data collection activities for wireless sensor network easier and convenient from user perspective. Few prototype sensor nodes were built to replicate a real life data acquisition network.

First objective was to develop a controlling application which would run the wireless sensor network in various mode of operation. It should also collect and save the data using a structured database management service.

Second objective was to develop a web framework and a few service-based web applications to serve both local and remote users.

1.3 Contribution of the Research

Wireless sensor network has a lot of applications. The developed system is highly suitable for environmental monitoring. Enthusiasts and hobbyists started using small pocket-sized computers like Raspberry Pi, Beagle Bone for their small home projects. Through my work I have shown it can be also used as a micro server for scientific research. The earlier web framework approaches for wireless sensor network had one common aspect, in most cases the gateway node and the webserver or the data storage server were two separate nodes. In my development I used Raspberry Pi both as the gateway and the webserver which makes the cost of implementation very low.

Through the HTML clients I have provided services like real time monitoring, database maintenance, configuring mode of operation, etc. All these can be done from a remote location which provides a lot of freedom for the research activity. I have followed a modular design strategy which made the web application independent of controlling application. So, if the web application has to go through any maintenance or upgrade, I can still run the system and access data through SSH client.

The system is highly portable both in terms of software and hardware. Raspberry Pi has very low power consumption comparing to a Personal Computer or legacy servers. So, we can actually deploy it in outdoor location without spending much for the battery backup. The web application and the controlling application were designed in Linux environment, so for mission critical application whole operation can be shifted to a proper industry grade server without consuming much time.

1.4 Organization of the Thesis

In Chapter 2 discusses the hardware that is used in the system. Choice of hardware defines the cost, development strategy, long term reliability and accuracy of the system.

Chapter 3 contains the software platform required for the development. Here I talk about the software or tool that are used in the system, also mention few other alternatives. I carefully chose an open source software base to reduce the cost of system development. This chapter gives a good idea about the required skill for the development of a web application for embedded system.

Chapter 4 deals with the basics of wireless sensor network. There are different protocols and devices available for short range wireless communication. This chapter describes the one I used and especially those features of the protocol which are essential for the development.

In Chapter 5 depicts different architecture of sensor network. I explain different scenario and introduce a method to solve data collection problem. Finally I propose the architecture that is implemented.

Chapter 6 is very important because from this chapter step by step the method of development is introduced. Various aspects of the design are discussed throughout the chapter.

In Chapter 7 introduces the final product of the development. I explaine how to operate it, demonstrate different mode of operation. I also introduced some data collection results.

Chapter 8 and 9 gives a summary of the entire thesis work and also talks about the future scope of work.

CHAPTER 2

HARDWARE PLATFORM

2.1 Introduction

This chapter discusses the hardware based on which the system was built. Whole system was based on a number of open source hardware and software so that the cost of building the system could be kept low. One of the goals while designing the system was to replace traditional personal computer in the system integration. There were quite a few options to choose from including the popular Beagle Bone [15] and Raspberry Pi [16] units. I chose the latter because it's economical and has tremendous support in the online community. Each of our sensor nodes was a combination of Arduino [17], Arduino XBee shield [18] and an XBee module [19]. An adapter kit [20] was used to connect the coordinator XBee to Raspberry pi. RHT03 [21] was used as a sample data acquisition sensor.

2.2 Raspberry Pi

Raspberry Pi is a credit card sized computer that can be used as a desktop PC. It is small but powerful enough to do regular word processing, spread sheet analysis and other regular activities. It can also do nonstandard function like being a media server or act as a smart TV. The original dimension of this machine is 3.37"x2.21"x.83". It was publicly launched on March 2012. There are two versions available: one with 256MB RAM known as Model A ($25) and the other with 512MB RAM known as Model B ($35). In this project Model B was used. Model A can also be used if lower power consumption is preferred but it lacks Ethernet port (not suitable for server). The graphics performance is like the original Xbox performance and CPU performance

is like a Pentium 2 300 MHz. The system does not come with a built in real time clock. Operating systems include any unix-like compiled for ARM architectures including linux and freeBSD. Monitor, keyboard and mouse can be connected with it and start using it as a desktop or it can be connected within a local area network using Ethernet cable or Wi-Fi adapter and then login using SSH client like putty.

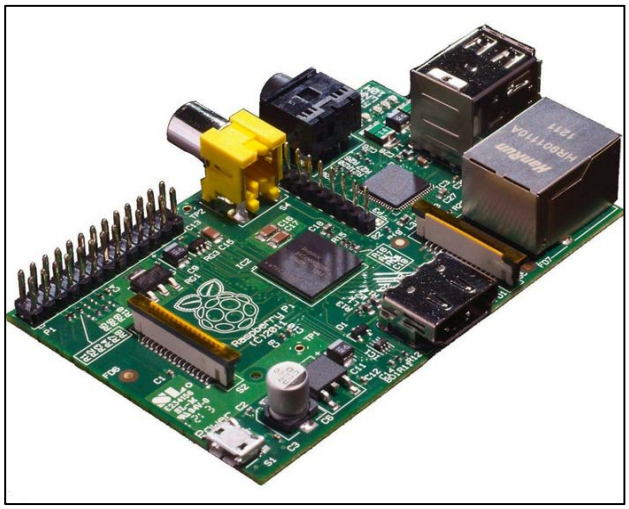


Figure 2.1: Raspberry Pi [33]

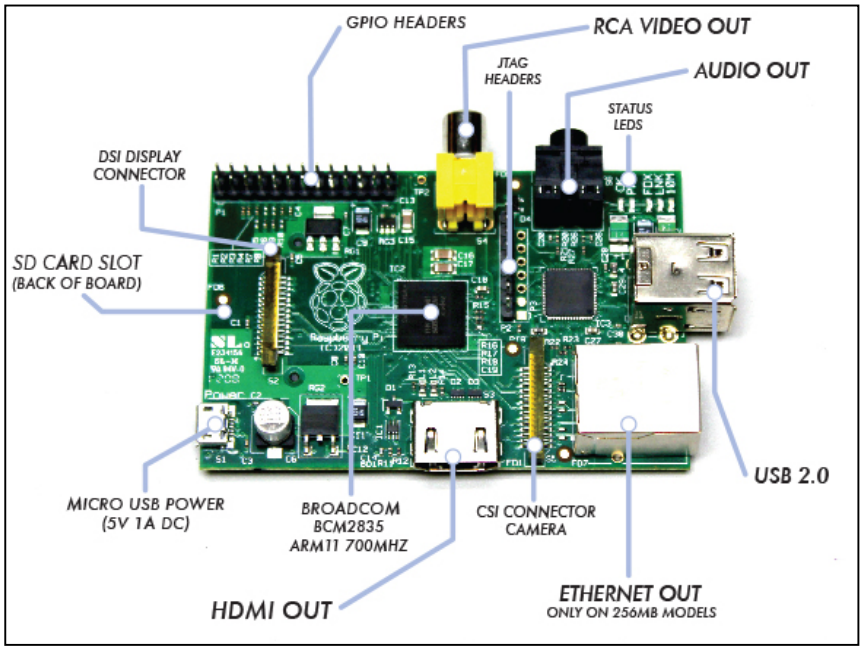


Figure 2.2: Raspberry Pi Ports and Connections [34]

One big advantage of Raspberry Pi over regular personal computer is that it has some easily accessible GPIO port which means it can be directly connected to buttons, motors, sensors, etc.

The detail specification of this board is given in the below table.

Table 2.1

Raspberry Pi Specification

Component	Type/Count
SoC	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, USBport)
CPU	700 MHz ARM1176JZF-S core (ARM11 family)
GPU	Broadcom VideoCore IV @ 250 MHz
Memory	512 MB (shared with GPU)
USB 2.0 ports	2
Video input	A CSI input connector
Video outputs	HDMI
Audio outputs	3.5mm jack, HDMI
Onboard storage	SD card
Onboard Network	10/100Mbps Ethernet (8P8C)
Low-level peripherals	8 × GPIO,UART, I²C bus, SPI bus with two chip selects, I²Saudio +3.3 V, +5 V, ground
Power ratings	700 mA (3.5 W)
Power source	5 volt via MicroUSB or GPIO header
Size	85.60 mm × 53.98 mm (3.370 in × 2.125 in)
Weight	45 g (1.6 oz)
Supported OS	Arch Linux ARM, Debian GNU/Linux, Gentoo, Fedora, FreeBSD,NetBSD, Raspbian OS, RISC OS, Slackware Linux

2.3 Arduino

Each sensor was built around an Arduino microcontroller, another example of open source hardware. The circuit design along with PCB layout is available online. There are quite a few versions of Arduino available, for example: Arduino Uno, Arduino Leonardo, Arduino Due,

Arduino Mega, etc. I used Arduino Uno in this development. Arduino is based on ATmega328. The package contains a 16 MHz ceramic resonator, a USB connection, a power jack and ICSP header and a reset button. Instead of using the FTDI USB-to-serial driver chip our Arduino features the Atmega16U2 chip programmed as a USB-to-serial converter.

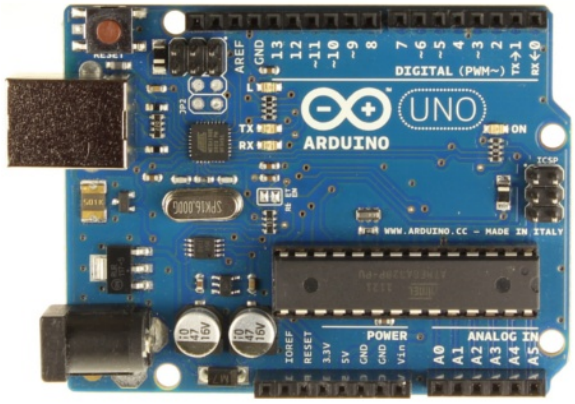


Figure 2.3: Arduino Uno R3 [17]

The detail specification for Arduino is provided in the following table.

Table 2.2

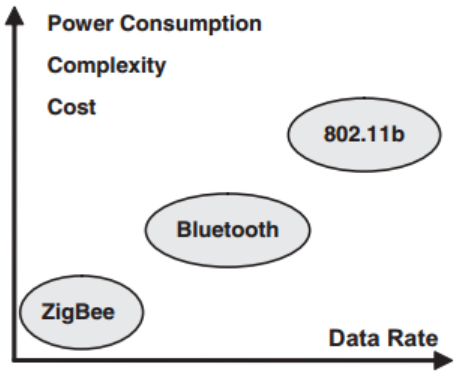
Arduino Specification [17]

Component	Type/Count
Microcontroller	**ATmega328**
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328)
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Arduino Uno can be powered by a USB connection, ac to dc adapter and battery. Each of the 14 digital pins on the Uno can be used as an input or output using functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor of 20-50 kOhms. In addition, some pins have specialized purpose. Pin 0 and 1 can be used for RX and TX TTL serial data. Pin 2 and 3 can be configured to trigger interrupt. Pin 3, 5, 6, 9, 10 and 11 can be used to provide 8 bit PWM output. The six analog inputs provide a 10 bits resolution.

2.4 XBee

For the wireless communication between sensor nodes and the gateway node ZigBee RF modules were used. All the ZigBee devices are based on ZigBee standard which has adopted IEEE 802.15.4 for its physical layer and MAC protocols. The wireless devices based on this standard operate in 868 MHz, 915 MHz and 2.4 GHz frequency bands having a maximum data rate 250Kbps.



	Data Rate	Typical Range	Application Examples
ZigBee	20 to 250 Kbps	10–100 m	Wireless Sensor Networks
Bluetooth	1 to 3 Mbps	2–10 m	Wireless Headset Wireless Mouse
IEEE 802.11b	1 to 11 Mbps	30–100 m	Wireless Internet Connection

Figure 2.4: Comparison between ZigBee, Bluetooth and IEEE 802.11b [3]

Bluetooth and Wi-Fi is the other two examples for short range wireless networking. Bluetooth has a typical data rate up to 3 Mbps with indoor range between 2 to 10m. Wi-Fi based on IEEE 802.11 family has typical range somewhere between 30 to 100 meters having

data rate up to 100 Mbps. Communication protocol for ZigBee has been kept simple compared to Wi-Fi to reduce power consumption at the expense of lower data rate but the typical indoor range is quite similar to Wi-Fi. So it addresses the need for very low cost implementation of low data rate wireless networks with ultra-low power consumption. The comparison between these three protocols is summarized in Figure 2.4.

There are quite a few variants of ZigBee module available from Digi. I used XBee-PRO S2B XBP24BZ7WIT-004. There are 2 version of this XBee available which are known as XBee pro series1 and series2. It is important to note that they don't work together. It's not possible to use both versions in the same network. All of our XBee used for the design of the system were XBee pro series2 model. There is no basic hardware difference between the gateway XBee and those deployed in sensor network. The difference lies mainly in their software configuration. Digi already has software named XCTU for configuring XBee, details of which is going to be discussed in later chapters.

Figure 2.5: XBee Pro S2B [19]

The detail specification for XBee is provided in the following table.

Table 2.3

XBee Pro Specification [19]

Specification	XBee Pro (S2B)
RF Data Rate	250 Kbps
Indoor Range	300 ft
Outdoor Range	2 miles
Configuration Method	API or AT
Interference Immunity	DSSS
Channel Access	CSMA-CA
Encryption	128-bit AES
Supply Voltage	2.7 - 3.6 V dc
Transmit Current	220 mA
Receive Current	62 mA
Power-Down Current	4 uA
Receiver sensitivity	-102 dBm

2.5 XBee Arduino Shield

To make a sensor node the microcontroller was connected with the wireless module using wire and breadboard according to Table 2.4. This can be inconvenient for deployment. There are various add-on boards available for Arduino commonly known as shield made by various third party companies. These add-ons can be attached vertically atop the Arduino board. Fortunately there is such a board available from Libelium which first attach itself to the Arduino board and then XBee is attached on top of it. It's possible to directly configure XBee using XCTU as long as the Arduino is connected to a PC.

Table 2.4

Direct Connection between XBee and Arduino

XBee	Arduino
VCC or 3.3 V	3V3
TX or DOUT	RX or 0
RX or DIN	TX or 1
GND	GND

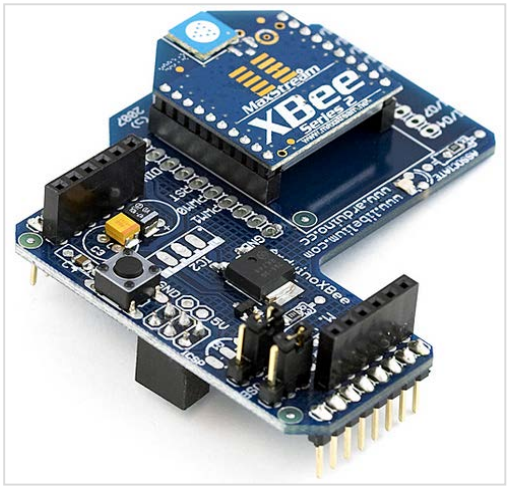


Figure 2.6: XBee with Shield [35]

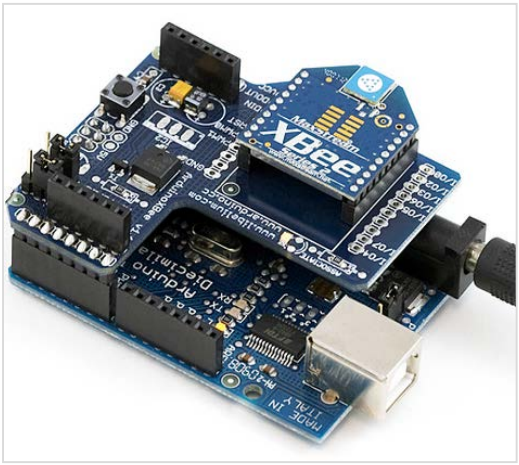


Figure 2.7: Arduino with XBee and Shield [36]

2.6 XBee Adapter Kit

To connect the gateway node with the Raspberry Pi I have used XBee adapter kit from Adafruit. Using this adapter kit and a FTDI cable we can directly connect the XBee module to any PC. It's a much more cost effective choice for the gateway node. The board has 10 port socket on both side so that XBee modules can be swapped very easily. Most of the common XBee pins have been drawn out to one side. So it can be easily connected to bread board or FTDI cable. There is a 3.3V on board regulator which provides ample current to support all version of XBee. Two LEDs are also available, one of them is for RSSI and the other one is for power.

Figure 2.8: XBee Adapter Kit [20]

2.7 RHT03 Sensor

A wide variety of sensors could be used with this network framework. To make the development initially simple I started with only one type of sensor. RHT03 was used to collect humidity and temperature data from environment. It's a low cost sensor having relatively high accuracy. Pin1 and Pin4 are connected to the VCC and GND respectively. Pin 2 can be connected to any of the digital IO port of the Arduino to collect data from the sensor. The

sensor is relatively slow in nature which takes about 2 second between each reading.

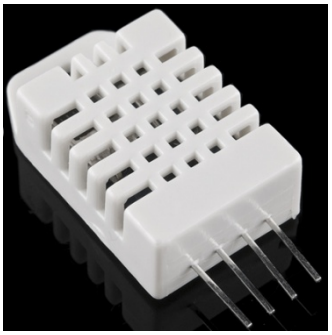


Figure 2.9: RHT03 Sensor [37]

Table 2.5

RHT03 Specification [21]

Component	Type/Count
Power supply	3.3 - 6V DC
Output signal	Digital Signal via MaxDetect 1 - wire bus
Sensing element	Polymer Humidity capacitor
Operating range	humidty 0-100%RH
Accuracy	Humidity +-2% RH
Max Current	1.5 mA
Standby Current	50 uA
Collecting period	2 s

CHAPTER 3

SOFTWARE PLATFORM

3.1 Introduction

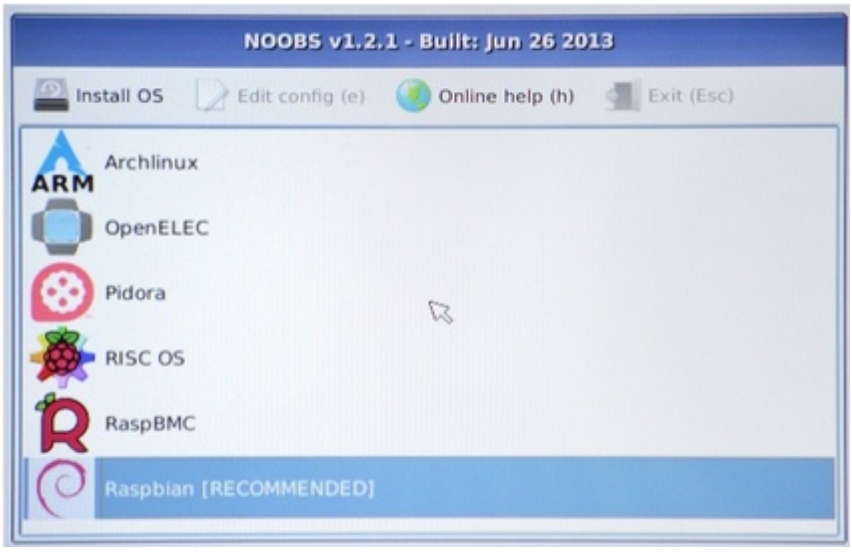
In this chapter I have discussed the software and modules that were used while developing the system. Raspberry Pi doesn't come with an operating system pre-installed so an appropriate operating system is selected for our need. Various software and tools to convert Raspberry Pi into a web server are introduced. One good aspect about the Linux distribution that is used is that it came built in with python as programming language. To be able to make it compatible with hardware and software various python libraries were used too. At the end I show the software module used to program Arduino and setup XBee.

3.2 Linux Distribution for Raspberry Pi

There is a common trend going on for small pocket sized computers to use Linux as the operating system. Even the android phones used as mobile device also runs on Linux based system. Linux is highly customizable as it is open source. There are quite a few distribution of Linux can be seen around us such as Red Hat, Fedora, SuSe, Mandrake, Debian, etc. Not all of these are light weight and suitable for Raspberry Pi. The Raspberry Pi organization has done a fantastic job by customizing the Linux which gets installed flawlessly onto the system without much prior technical knowledge. They have introduced a package known as NOOBS (new out of box software) which can be downloaded from their website. I copied the content of the NOOBS on the SD card coming with Raspberry Pi or and then connected it with the power supply which boots the system immediately. There are quite a few online materials [22][23] which describes

the detail procedure for OS installation. At the beginning of the installation it gives quite a few options from where I can chose the distribution which is suitable for building the system. From various resources [24][25] it was found that Raspbian got highest recommendation. It gives an all-round OS experience which is very easy to use for a beginner. Raspbian has a basic LXDE environment and is derived from Debian Wheezy. Pidora is derived from Fedora optimized for Raspberry Pi. Both RaspBMC and OpenELEC are derived from XBMC media center.

Figure 3.1: NOOBS Choice of OS for Raspberry Pi [23]



At some point during the installation a dialog box named Raspi-config appears. Raspi-config allows to select some important options for the operating system before actually booting it up.

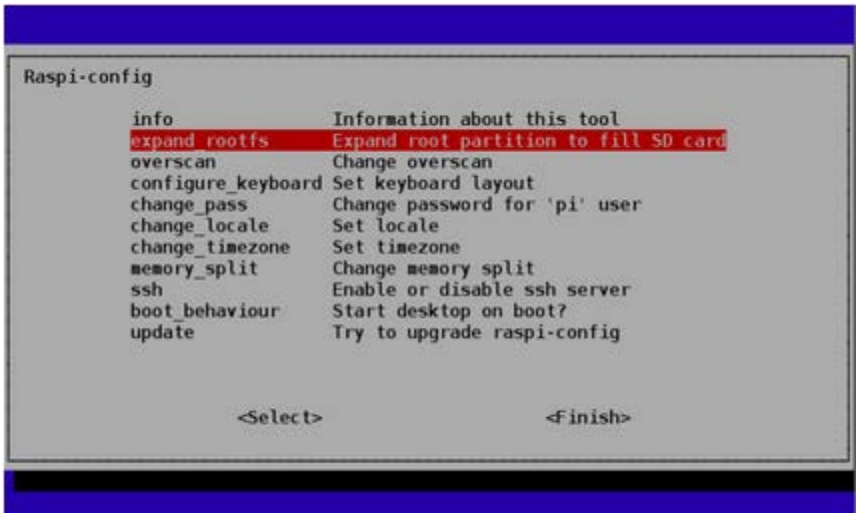


Figure 3.2: Raspi-config [23]

- Expand_rootfs: One typical characteristic of the default Raspbian distribution is it only uses as much space as the operating system requires even though the size of the SD card being much larger. To solve this problem we can select the option expand_rootfs which will allow the operating system to expand its file system beyond its size and occupy rest of the unused space of the SD card.
- Overscan: Raspberry Pi can be connected to any monitor or TV. Sometimes it is found that the screen occupies only middle portion of the monitor or TV. To get rid of this problem Overscan from raspi-config can be disabled.
- Configure_Keyboard: This gives the option to select the desired keyboard layout. It is important because the default OS comes with the UK version of keyboard layout which is slightly different and inconvenient from the version that is normally used.

- Change_pass: The default user name for the system is “pi” and the password is “raspberry”. From the change_pass option of Raspi-config user can easily change the password at the beginning. This is important for the security issue of the server.
- Change_timezone: From this option actual time-zone can be configured.
- SSH: Using Secure Shell or alternatively known as SSH user can remotely login to the server. This is important because I wanted to remotely change configuration of the system, change, modify or upload codes. SSH is a secure option to do that. I can use Putty, Secure CRT or any other similar software to remotely login to Raspberry Pi. Using this option from Raspi-config I can select whether it is required to enable SSH or not.
- Boot_behaviour: Raspberry pi boots with a default command line option. For the beginner it is better to use the desktop environment. From this option user can change that so that it boots straight to desktop every time it boots up.

It is important to mention that Raspberry Pi doesn’t have any dedicated power or reset button. It boots automatically whenever it’s powered up.

3.3 Converting Raspberry Pi to Web Platform

Webserver is a combination of hardware and software delivering web content which can be accessed by clients or users through internet or local area network. The communication between the server and the client takes place using Hypertext Terminal Protocol. Any general purpose computer should be enough to be the hardware platform, in this case which is Raspberry Pi. To handle http requests from the client I also needed software. The website or the html pages can be designed anywhere but to access them from web or LAN I had to save

them to a specific location inside the server. So, whenever http requests were sent from the client side the webserver responded by forwarding corresponding html contents to the client. In the following sections I will show the software that is used to build our webserver.

3.2.1 Webserver

Sometimes webserver only refer to the software that handles http request. There are quite a few options like Apache, Cherokee, Lighttpd, Nginx, etc for Linux environment and IIS for Windows. All of them except IIS are open source and free of cost. The performance of a webserver depends on how many clients it can handle simultaneously and how fast it can response. Nignx is the fastest web server seen around in terms of load time and number of connections and Apache is the slowest one [26]. Apache is significantly slower than Nginx especially for static pages. In the case of dynamic pages Apache is not that slow. Most of my pages are full of dynamic content. Moreover around 50% of the total webserver in the world use apache and I took the same route. Two versions of apache are available known as apache1 and apache2, I used apache2.

I have mentioned before that html pages can be designed and viewed anywhere. When these are required to be served someone have to make sure they reside in proper location within the webserver. As soon as the installation is done a new directory named “www” is created under the directory “var” which is also known as document root. When a certain page is requested apache searches whether that page is available within the document root, if not user will not get the proper response from apache. If a different directory is required within the server in that case apache has to be configured accordingly.

3.2.2 Database Engine

The received data from the sensor network can be directly saved in a text file. To access those data properly or to manipulate it is required to store them in a more structured manner. That's why I have considered installing a database engine capable of serving the purpose. There are several database engines like MySQL [27], SQLite, PostgreSQL. All of them are open source and are free to use. I chose MySQL as it is very highly ranked and most widely used RDBMS. SQLite could also be used considering its small footprint. I did some experiment during the initial stages of the development with sqlite3. During the installation I was prompted to use a password for MySQL. This was important because every time it is required to access MySQL directly from a terminal it prompts for user authentication. Similarly scripts that are written to access the database will require proper authentication.

I had to use two modules for MySQL which helped it to function properly.

- Libapache2-mod-auth-mysql: To do HTTP authentication by apache2 against information stored in a MySQL database I installed the module "libapache2-mod-auth-mysql".
- Php5-mysql: I can access database from any terminal. To connect with the MySQL database directly from PHP script it is required to install the module php5-mysql.

3.2.3 Server Side Scripting

From the HTML page If it is required to directly log in to the database and retrieve or modify data; we need to have a program residing in the server which is responsible to communicate with the database. The script can stay in the server as a separate file or can be

embedded in the html code. The reason this type of programming known as server side scripting because they are executed by the server. A scripting language helps to design dynamic webpage. There are quite a few scripting languages available for example: PHP [28], ASP, JSP, Ruby, Python, etc. I used PHP for the development. The reason is it's free, requires less processing power in server end, suitable for both small and big projects, requires small development time and overall its most widely used by the web programmer.

I also installed the tool phpmyadmin [29] which helped to administrate MySQL. Using this option to graphically work with MySQL is possible and do most of the functional task. I can create, copy, drop, rename and alter databases, tables, fields and indexes. User can import data from CSV and SQL to the database and also can export any data from MySQL to CSV or SQL format using phpmyadmin.

3.4 Charting Tool

One of the goals of the development was to save the data and also to plot it as a real time graph or chart. Using JavaScript I could build our own charting tool. It proved to be difficult and time consuming. I followed an alternative route. There are various charting tool available online which are based on JavaScript and jQuery and some of them are free. Such as Flot [30], JS Charts, PlotKit, ProtoChart, EJSChart, etc. Any of them can be chosen and then embed the scripts to our main html coding to plot the required chart. In this development I used flot chart. The libraries for JQuery and Flot were kept in the document root folder. In the following sections there is more discussion on JQuery and Flot Chart.

3.4.1 JQuery and Ajax

JQuery [31] is a JavaScript library using which can be used to do client side scripting in HTML much faster than the original JavaScript. It is free and open source software. This can be used to do document navigation, animation, event handling in less number of lines of codes. Any kind of charting or plotting tool can be used but without data there will be nothing to show. JQuery is used in this development to feed the charting tool with continuous flow of data from the MySQL database via the server side script. JQuery in tandem with Ajax [32] sends an asynchronous data request to the server and only updates the data part of the chart with the received data not the whole html page.

3.4.2 Flot Chart

Flot is a JavaScript plotting library for jQuery. It is simple to use with some attractive looks, interactive features and excellent browser support. I used version 0.8.2 which supports Explorer 6+, Chrome, Firefox 2+, Safari 3+ and Opera 9.5+. It has quite a few interesting options for example, the legend, axes, grid, data series, time series, use gradient, etc can be customized. The input for the Flot chart is an array of data series and the data has to be always a number. Flot won't be able to handle the data which is forwarded as string format. While handling time series data the plugin jquery.flot.time.js should also be used to support the time series. The time series support in Flot is based on JavaScript timestamps, For every time value a JavaScript timestamp number is used. A JavaScript timestamp is the number of milliseconds since January 1, 1970 00:00:00 UTC. This is almost the same as Unix timestamps, except it's in milliseconds, so to get the second I have to multiply by 1000.

3.5 Python Modules and Packages

Python comes built in with Raspbian, so there is no need of installation. In this development I used python to build the controller software which would control the sensor network, send and collect data from the sensor network and save them in a database. Python has many built in functions and modules. In addition to that I had to install few more packages or modules to serve the purpose.

When our program gets longer it is possible to take a block of definitions and put them in a separate file. This file can be used by other programs too. In python by stacking set of functions together it is possible to create a separate reusable file which can be referred as module. In other words module is a file containing Python definitions and statements with a suffix .py. A package is a collection of modules. By importing the package it is possible to access all the modules and sub modules within its scope.

Below are some of the important packages that are used in this development.

- Pip 1.5.2: This is the first package for python that should be installed in a in a brand new Linux machine. It helps to install and manage python packages.
- PySerial 2.7: Most of the data will be written through serial port or will be read through it. Setting up the pySerial module enables python to access the serial port.
- MySQL-python 1.2.5: Python cannot communicate directly to the MySQL database. Whether it is required to read or write data from a MySQL database using python, it is required to install the module MySQL-python.
- Advanced python scheduler: APScheduler 2.1.2 is a powerful task scheduler which helps to schedule jobs to be executed at a chosen time. This is used in this project. It's an

alternative to cron but the advantage is it's a in-process task scheduler. It is platform neutral and can directly access application variables and functions.

- Xbee-2.1.0: This package provides an implementation of the XBee serial communication API. Features in XBee devices can be accessed from an application written in Python by using this package. It cannot work without Pyserial.

3.6 Arduino IDE

Any micro controller that is seen around requires a development environment where it can be programmed before actual deployment. Fortunately the maker of arduino provides an open source IDE that can be downloaded for free. Current version of this IDE is Arduino 1.0.5. It hides the messy details of microcontroller programming and provides an easy to use programming environment for beginners. Programming syntax is similar to C. Another good thing about this IDE is it can be used in cross platform environment supporting windows, Linux and mac os. I can add C++ libraries and enhance more programming options. The IDE comes with a built in serial monitor which proves to be very much useful for debugging.

For a brand new Arduino board it requires to install the driver otherwise the host pc wouldn't identify the hardware. I can write our program on the GUI and then upload the program into the board. Before uploading the program I had to select the serial port correctly. While doing the microcontroller programming I used two libraries following is the brief description of those libraries.

DHT: This library is developed by Mark Ruys which makes the use of RHT03 sensor very convenient for us. It supports DHT11, DHT22, AM2302 and RHT03. The library has low memory footprint and requires very small amount of code to read the sensor values.

XBee: It is used to do the task of communication between XBee's in API mode. ZigBee has several type of Frames and this library supports most of them. It took out the complexity of building frames in api mode and also helped to send and receive packets in a much more convenient way using built in functions.

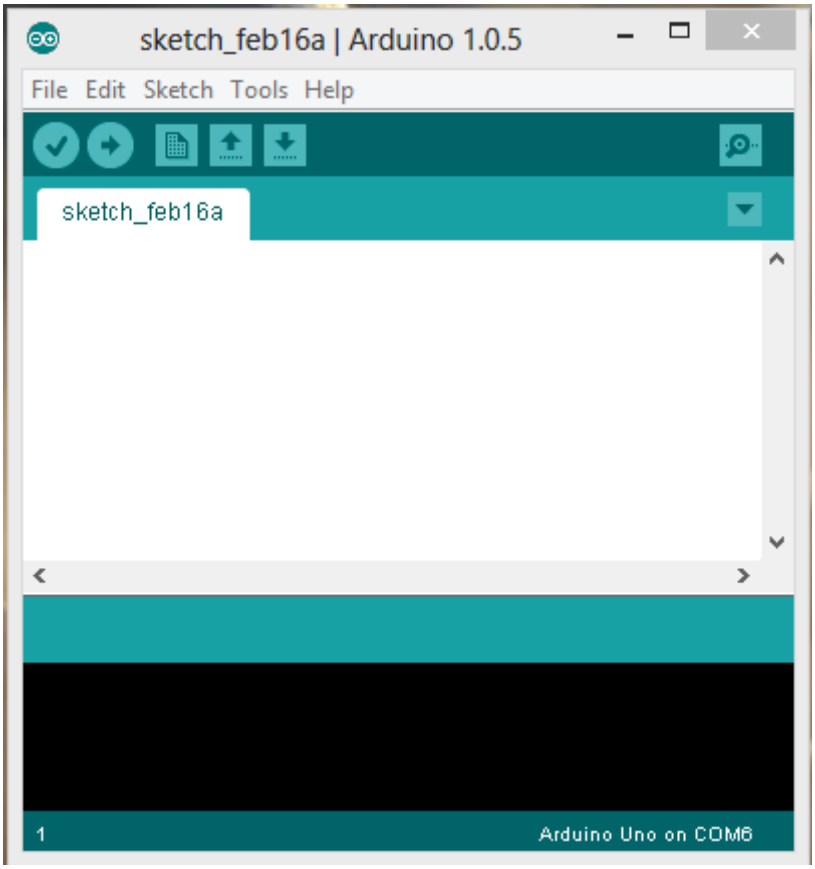


Figure 3.3: Arduino IDE

3.7 XCTU

Before using any XBee module it is required to set some parametersfor each of the modules. There is a nice software named XCTU from digi international which helped to set these parameter. I installed the software in a separate windows machine and did all the configuration from there. Using the adapter kit and the FTDI cable it is possible to connect the XBee module to any computer.

PC settings of XCTU is used to test the connectivity of XBee with the computer. The baud rate can also be set from here. Terminal window in XCTU helped to see incoming and outgoing packet both in character and hex mode. Using the window Modem configuration parametrs for XBee module can be configured.

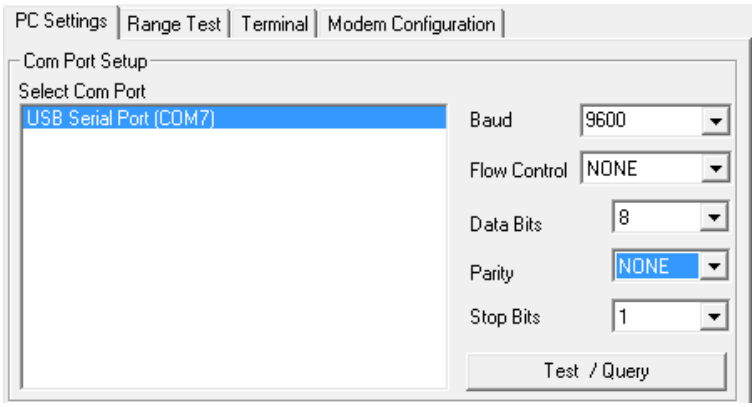


Figure3.4: XCTU Software

CHAPTER 4

ZIGBEE NETWORKING AND PROTOCOLS

4.1 Introduction

Before going deep in to the system development I want to have a better understanding on this wireless technology. The XBee module used here operates at 2.4 GHz. At this band ZigBee standard has 16 channels within 2400 – 2483.5 MHz. The modulation scheme is Orthogonal QPSK. The chip rate, bit rate and symbol rate are 2000 Kchip/s, 250 Kb/s and 62.5 Ksymbol/s respectively.

4.2 ZigBee Protocol Stack

ZigBee protocol layers are based on OSI model. When the pan is to use ZigBee it is necessary to mention IEEE 802.15.4 standard. One of the finest characteristics about this standard is it allows user to use PHY and MAC layer defined by IEEE 802.15.4 and lets user to define the upper layers of the OSI model. Similarly ZigBee also use the MAC and PHY layer of IEEE 802.15.14 standard.

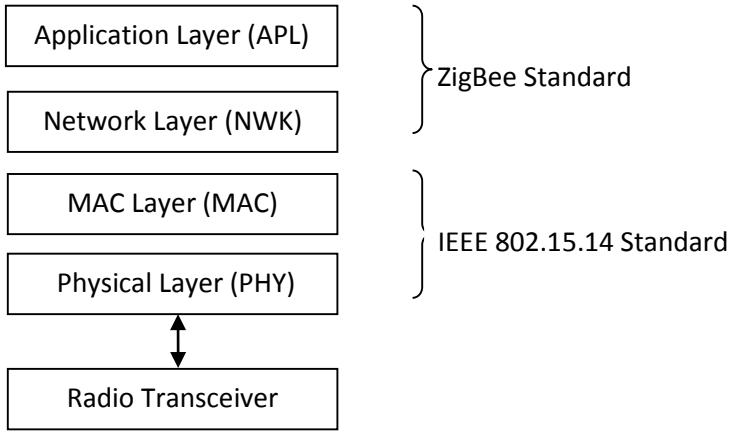


Figure 4.1: ZigBee Protocol Stack

PHY layer is the closest layer to hardware and is responsible for activation of the radio that transmits or receives packets. This layer also selects the channel which is currently unused. PHY layer directly controls and communicates with the radio transceiver. PHY packet contains three components: Synchronization Header (SHR), PHY Header (PHR) and the PHY payload.

MAC is the layer in between NWK and PHY layer which is responsible for generating beacons, synchronizing the device with the beacon, provide association and disassociation services. Each MAC frame has three sections- MAC Header (MHR), MAC payload, MAC Footer (MFR). There are four types of MAC frames defined by IEE 802.15.4.

- Beacon Frame: Synchronizes the clock of all the wireless devices within the network
- Data Frame: Used for transmitting data
- Acknowledge Frame: is used to acknowledge after successful reception of a frame
- MAC Frame: MAC commands are transmitted using this frame.

The Network layer stacks in between Application layer and Mac layer and is responsible for network formation and routing. Routing means selecting a path using which message gets delivered from source to destination. In a typical ZigBee network Coordinator and router devices performs route discovery and maintains the routes in the network. NWK layer of the ZigBee coordinator establishes a new network, selects topology and provides network addresses to other devices within the network. NWK frame has two parts: NWK header (NHR) and NWK payload.

Application layer is the top layer in the protocol stack responsible for holding application objects. The purpose of the application object is to control and manage protocol layers in a

ZigBee device. While developing an application this standard also has the option to use application profile which is nothing but a set of agreements on application-specific message formats and processing actions. Products from two different vendors interact with each other if they were developed by using the same application profile. That's how interoperability is achieved.

4.3 Device Types and Roles

According to IEEE 802.15.4 there are two types of devices under this standard: full function devices (FFDs) and reduced-function devices (RFDs). FFD device can communicate with any other device in the network whereas RFD device can communicate only with an FFD device. In ZigBee standard we have three types of devices, ZigBee Coordinator, ZigBee Router and ZigBee End device. Coordinator is an FFD type device having the controlling ability of the personal area network. Router is also an FFD type having the ability to relay messages. End device has the least capabilities which can only communicate with the ZigBee Coordinator or Router. There can be only one coordinator in a network. Number of router or end device is not limited. Since the coordinator is subject to heavy communication load normally its connected to mains power.

4.3 Networking Topologies

ZigBee networking layer (NWK) manages the network formation. The network should be either star connected or peer to peer connected. When all routers and end devices in a network directly communicate to the coordinator it is known as star connection. For any device to be able to communicate with the coordinator it has to be within the radio sphere of influence of

the coordinator. This is a problem for the star connected network because it is not possible to grow the network beyond the coverage of coordinator.

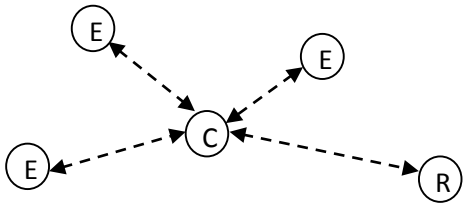


Figure 4.2: Star Connected Network

Generally in peer to peer communication every node is able to communicate with every other node. There would be one coordinator and rest of the node should be able to relay message which means they have to be router. This type of peer to peer topology is known as mesh network where everyone talks with everyone.

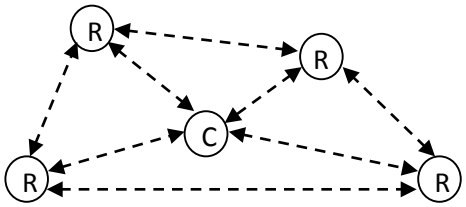


Figure 4.3: Mesh Connected Network

Another type of peer to peer network is supported by ZigBee known as tree topology. First of all coordinator establishes the initial network. Routers form the branches and relay the messages. The end devices act as leaves of the tree and do not participate in message routing. Router can grow the network beyond the initial network established by the coordinator.

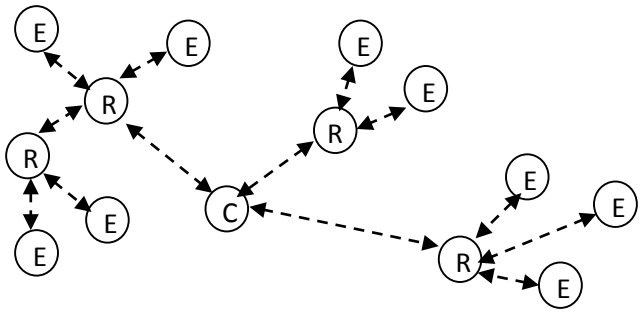


Figure 4.4: Tree Connected Network

No matter which topology is used coordinator controls the network and does the following minimum tasks:

1. Allocate a unique address (16-bit or 64-bit) to each device in the network
2. Initiate, terminate and route the messages throughout the network
3. Select a unique PAN for the network.

4.4 ZigBee Parameters

In this section I introduce important ZigBee parameters. Most of these parameters can be configured through XCTU.

- Function Set: From this option user define the device role and select mode of operation.
- BD: Baud Rate is the speed of the serial interface which should be same for all the devices.
- PAN ID: All the devices within a network must have the same PAN ID.
- SH and SL: Every XBee device has a unique 64-bit address. This parameter gives the source address.
- DH and DL: When user want to send data to another XBee the 64-bit address of the receiver should be set to DH and DL of the sender so that sender gets the information regarding where to send data.
- MY: This parameter is known as 16-bit Network Address which is automatically assigned by the coordinator.
- PL: This is Power Level, using this parameter it's possible to select the transmit power level of the device.

4.6 Serial Interface Protocols

The XBee module has support for transparent (AT mode) and Application Programming Interface (API mode) serial interfaces. Below is a short description for both modes.

AT mode: In this mode the XBee module behaves as a serial line replacement. All the UART data received from the microcontroller is queued up for RF transmission. When data is received it is sent out through the DOUT pin. This is the simplistic version of serial communication using XBee modules.

API mode: In this mode all data entering and leaving the module is contained in frames that define operations or events within the module. Using this mode it is possible configure module and routing data at the host application layer. Transmitting RF data to multiple remotes only requires changing the address in the API frame. This process is much faster than in transparent operation where the application must enter AT command mode, change the address, exit command mode, and then transmit data. Each API transmission can return a transmit status frame indicating the success or reason for failure. Some of the advantage of this mode is mentioned below:

- Supports both broadcast and multicast transmission
- Packet include checksum for data integrity
- RSSI can be obtained from a RX packet
- Remote radio can be configured with the remote AT feature.
- Received packets contain source address.

4.7 ZigBee API Frames

When it is required to communicate with the wireless modules using API operation, data is sent and received in a defined structure. Each of these structures is implemented as UART data frame. There are 2 different versions of API mode available known as AP1 and AP2. Following is the UART Data Frame Structure for AP1:

Figure 4.5: ZigBee API Frame Structure

Here 0x7E is known as the start delimiter which signifies the start of a new frame. If user finds the same delimiter anywhere else in rest of the bytes of the frame it's possible to get confused. To avoid this collision AP2 is introduced where bytes after the delimiter byte is escaped if needed. The bytes that need to be escaped are 0x7E, 0x7D, 0x11 and 0x13. To escape a character it is required to introduce a character 0x7D and then XOR the original character with 0x20.

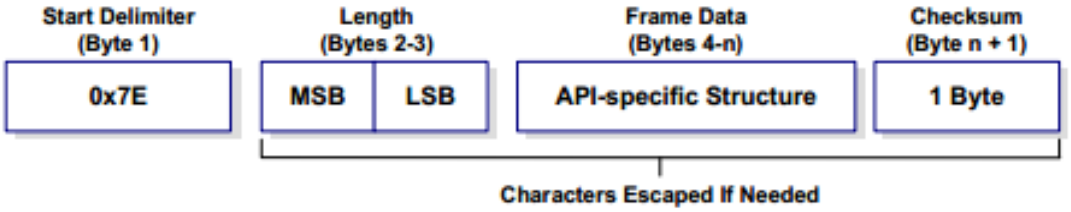


Figure 4.6: Characters Escaped in API Frame

Next two bytes are used to specify the size of the frame which equals to the length starting from Byte after the length till the end of frame data. The last byte known as checksum is used to check the integrity of the data. To calculate the checksum I sum all the bytes of the payload and deduct the sum from 0xFF. The content of Frame data varies depending upon the

payload and also the type of frame I want to send. A raw UART data frame is presented below for analysis: 0x7E 0x00 0x02 0x23 0x11 0xCB.

In the above example 0x7E is the start delimiter which signifies the beginning of the frame. The next two bytes gives us the size of the payload. So, the check sum is 0xFF – (0x23+0x11) = 0xCB. Now inside the payload we have 0x11, so in escaped mode the frame would be: 0x7E 0x00 0x02 0x23 0x7D 0x31 0xCB.

4.7.1 Frame Data

ZigBee has defined several application specific frame structures. General structure is same; the difference lies mainly in Frame Data. Each application has its own API identifier which is used as the start byte of the Frame Data. Following is the most commonly used API Frame names and API ID.

Table 4.1

List of Commonly Used API Frames. [1] [38]

API Frame Names	API ID
AT Command(immediate)	0x08
AT Command(queued)	0x09
AT command response	0x88
Modem Status	0x8A
ZigBee Transmit Request	0x10
ZigBee Transmit Status	0x8B
ZigBee Receive Packet	0x90
Remote Command Request	0x17
Remote Command Response	0x97
Remote Command Request	0x17

4.7.2 Frames Used for the Development

Here we see which type of frame requires to be used while designing the system and have in depth look at their Frame Data. I used following type of frames in our development:

- ZigBee Transmit Request
- ZigBee Transmit Status
- ZigBee Receive Packet

4.7.3 ZigBee Transmit Request

In Table 4.2 there is an example of ZigBee Transmit Request frame. Byte 1, 2, 3 and the last byte follows the general frame structure. Frame data starts with the API id reserved for TX request which is 0x10. Byte 5 represents Frame ID which is necessary to correlate frames between sender and receiver. Bytes 6 to 13 and 14 to 15 are reserved for 64bit destination address and 16 bit network address. If the sender wants to send data to coordinator in that case 0x0000000000000000 can be used as 64 bit destination address and the message is going to still reach the coordinator. If the coordinator wants to broadcast message to all the nodes in that case 0x000000000000FFFF can be used and message can be broadcasted to the nodes. Similarly if 16-bit address is not known 0xFFFE can be used. Broadcast radius at byte 16 specifies the number of hops the message should travel. If I use 0x00 its message travels maximum number of hops. Options in byte 17 enable us to implement various transmission modes like disable ACK, APS encryption, extended transmission, etc. RF data can be of variable size which is our actual payload. So depending upon this I have to calculate the data length for the Frame. The last byte is reserved for checksum.

Table 4.2

ZigBee Transmit Request Frame Structure [1]

Frame Fields		Byte no.	Byte value	Description
Start delimiter		1	0x7E	
Length		2	0x00	Number of bytes in Frame data
		3	0x11	
Frame data	Frame type	4	0x10	specifies the api type
	Frame ID	5	0x01	
	64-bit destination address	6	0x00	-64 bit long address of the destination -0x0000000000000000: reserved for coordinator -0x000000000000FFFF: Broadcast address
		7	0x13	
		8	0xA2	
		9	0x00	
		10	0x40	
		11	0x0A	
		12	0x01	
		13	0x27	
	16-bit destination network address	14	0xFF	- 16 bit network address for destination - 0xFFFE used when address is unknown
		15	0xFE	
	Broadcast radius	16	0x00	number of hops, 0x00 means maximum hop
	Options	17	0x00	transmission option
	RF data	18	0x31	Actual payload
		19	0x31	
		20	0x31	
Checksum		21	B9	

4.7.4 ZigBee Transmit Status

Advantage of using API mode is that for every transmission user can have the status whether message was delivered properly to the intended receiver provided Frame ID in original TX frame should not be 0x00. User receives a full status report known as ZigBee Transmit Status containing discovery, transmission status, retries, etc. The length of TX response is 11 byte.

Table 4.3

ZigBee Transmit Status Frame Structure [1]

Frame Fields		Byte Position	Byte value	Description
Start delimiter		1	0x7E	
Length		2	0x00	Number of bytes in Frame data
		3	0x11	
Frame data	Frame Type	4	0x8B	api id for TX response
	Frame ID	5	0x01	same id used for original TX request
	16-bit destination network address	6	0x7D	16 bit network address of the node where the packet was delivered.
		7	0x84	
	Transmission retry	8	0x00	increases with the number of retry
	Delivery status	9	0x01	
	Discovery status	10	0x01	
Checksum		11	0x71	

The byte discovery status gives the information about how much network overhead it took to discover the route for this transmission. The smaller number indicates lower network overhead. This frame is the response of the receiver to the sender after receiving transmission from the sender and delivery status 0x00 means data has been successfully delivered from

sender to receiver. If the delivery status is not 0x00 then the delivery is failed and each number is going to indicate a reason of failure.

Table 4.4

Discovery Status Options [1]

Discovery Status
0x00 = No discovery overhead
0x01 = Address discovery
0x02 = Route discovery
0x03 = Address and route
0x40 = Extended timeout discovery

Table 4.5

Delivery Status Options [1]

Delivery Status	
0x00 = Success	0x26 = BC source failed to hear a NB relay the msg
0x01 = MAC ACK failure	0x2B = Invalid binding table address
0x02 = CCA failure	0x2C = Resource error, lack of free buffers, timers, etc
0x15 = Invalid destination endpoint	0x2D = Attempted broadcast with APS transmission
0x21 = Network ACK failure	0x2E = Attempted unicast with APS transmission, but EE =0
0x22 = Not joined to network	0x32 = Resource error, lack of free buffers, timers, etc.
0x23 = Self-addressed	0x25 = Route not found
0x24 = Address not found	0x75 = Indirect message unrequested

4.7.5 ZigBee Receive Packet

After receiving an API packet from the sender, the receiver module requires to do two things. First it sends TX response to the sender acknowledging that it has got data. Then using the RF packet it sends out UART data to the microcontroller using the Frame Type 0x90. This

type of packet is known as ZigBee Receive Packet. It contains the 64-bit source address, 16-bit source network address along with the original payload.

Table 4.6

Frame Structure for ZigBee Receive Packet [1]

Frame Fields		Byte Position	Byte value	Description
Start delimiter		1	0x7E	
Length		2	0x00	Number of bytes in Frame data
		3	0x0A	
Frame data	Frame type	4	0x90	specifies the api type
	64-bit destination address	5	0x00	64 bit address of sender
		6	0x13	
		7	0xA2	
		8	0x00	
		9	0x40	
		10	0x52	
		11	0x2B	
		12	0xAA	
	16-bit destination network address	13	0xFF	16 bit address of sender
		14	0xFE	
	Receive options	15	0x01	0x01- packet acknowledged
	RF data	17	0x31	Actual payload
		18	0x31	
		19	0x31	
Checksum		20	C3	

CHAPTER 5

SYSTEM ARCHITECTURE

5.1 Introduction

In this chapter I have focused on the system architecture. While thinking of designing a data acquisition system; a simple approach may serve the purpose which can be proved cost effective both in terms of time and money. Here I will correlate between the various requirements and the corresponding system architecture. This ultimately led to the development of the system that is desired.

5.2 Type of System

Number of sensor nodes, the location where they need to be used to collect data, requirement for real-time monitoring, data storage facilities are the major factors defining the system architecture. Let's assume it is required to measure some environmental parameter using sensors. Sensor nodes can be built using the necessary sensors and microcontrollers and then I can program the microcontroller to get the sensor values. If the placement of the sensor node is within the proximity of the base station or host computer these can be easily connected using a serial cable and receive the data at regular interval. This is quite simple, cost effective and straight forward approach but things get interesting when the placement of the sensor node is far from the base station or for the cases when it is required to use them in outdoor location. To solve this problem the sensor nodes can be designed with storage service. Data is read by the microcontroller and logged simultaneously in the storage media. Numbers of sensor nodes can be built like this. The second option to collect data from a remote location is

to use a radio transceiver module for the sensor nodes and transmit data wirelessly to the base station where another radio transceiver will receive those data and save them in base station. This is known as wireless sensor network which is a smart and efficient way to collect environmental data. Based on the above discussion I can think of three types of system:

1. Wired sensor network
2. Standalone data logger
3. Wireless sensor network

In the following sections I discuss more on the above system architectures, introduce basic block diagram.

5.3 Wired Sensor Network

I have already discussed about this. A microcontroller can be used along with necessary sensors on top of it using appropriate circuit. The microcontroller can be directly connected to base station or host computer using RS232 serial cable or USB cable which opens a com port in the host computer. If it is required to connect multiple nodes like this each of them open individual com port. Then it is required to write a program which individually collect data for each sensor node and send those data to the serial port of the host computer. So each of these nodes is a standalone system and has no relationship between them. A program is required in the base station which is going to grab the serial data and save them in text or csv file. If real time monitoring is required then a GUI interface can be designed.

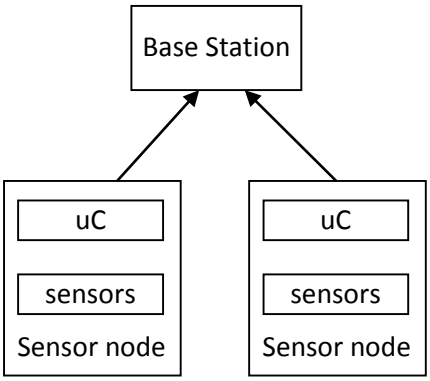


Figure 5.1: Wired Sensor Network

Advantage of this system is its very simple and cost effective. Power is drawn from the USB cable, so don't need to think about battery backup. The disadvantage is sensor node has to be within the reach of the base station which may not be practical always. The number of available serial port or USB port limits the number of sensor nodes that can be connected to the base station.

5.4 Standalone Data Logger

This concept can be easily implemented using Arduino. Arduino platform has broad support for data logging using a micro SD card. When it is not possible to connect a sensor node through wire or when it is required to keep it in a harsh outdoor environment where it is not possible to have the base station anywhere near it; standalone SD data logger can be a solution. I have seen a micro SD shield available from Sparkfun [13] which can be easily mounted on top of the Arduino Uno board. There is also a C++ library available which helps to write the collected data on the SD card.

There are some disadvantages of this process. It is not possible to start or stop the sensor node from a remote position; also not possible to see what's going on in the middle. Each time user needs data, someone have to go there and manually take data from the SD card.

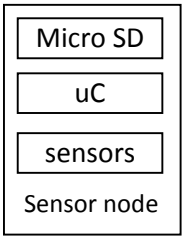


Figure 5.2: Standalone Data Logger

When sensor node is too far from the base station or the environment is very harsh, it is possible to connect GSM or 3G modem with the sensor node and periodically upload data to a webserver or cloud server. Arduino has tremendous support in this area too as both GSM [14] and 3G [15] shields are available. The cost of these GSM or 3G shield is quite high. So in case of large array of sensors the cost of the development will greatly increase. A large batter back up is required as these shields use more power.

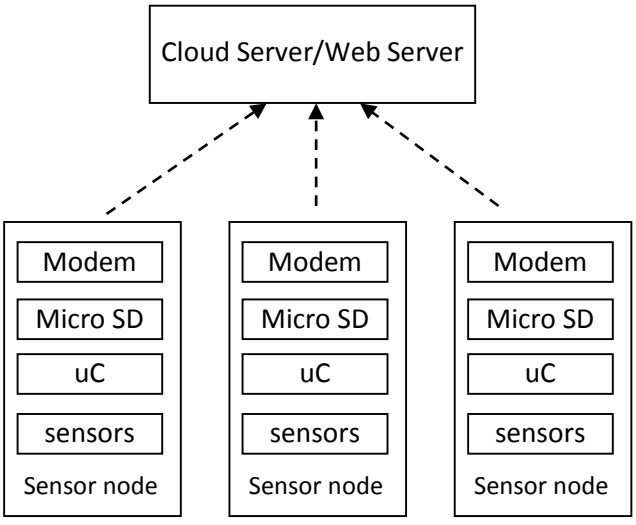


Figure 5.3: Modem Based Data Logger

5.5 Wireless sensor network

Wireless sensor network is a smart solution for the environmental data collection problem. Low power XBee modules can be used for wireless connectivity which makes it possible to form a mesh or tree network within the array of sensors and then transmit data to the gateway node. The Xbee connected to the base station is the coordinator. The XBee connected to the sensor nodes can be either router or end devices.

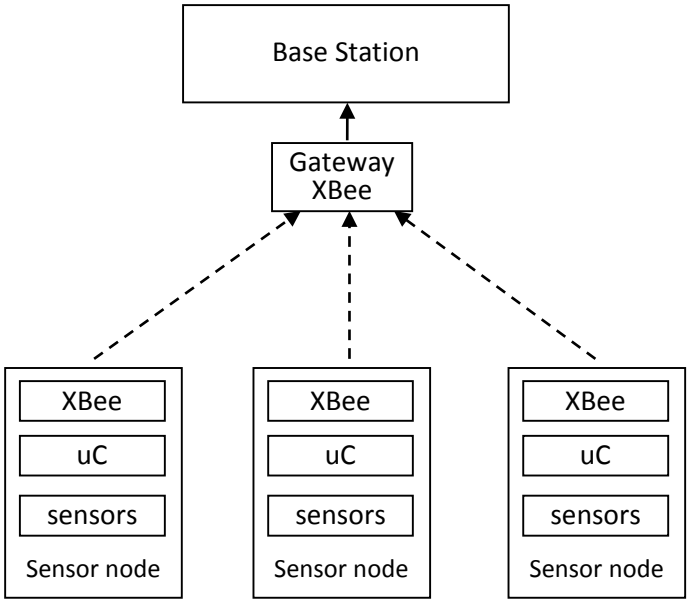


Figure 5.4: Wireless Sensor Network

XBee network can be configured either in AT mode or API mode. AT mode is very simple to implement but there is only Broadcast and Unicast option. To have the Multicast option user has to implement the API mode. On top of that it also gives more information, like source address, RSSI, number of hop. So, user can choose mode based on the requirement.

Data is available at gateway node, now it can be saved in a text file at the host computer by using a serial reader program. It can also be saved in a structured way by using a database. Now if it is required to remotely access the data user can upload the data from the base station to a webserver or can use the base station as a webserver. If the concern is just to get the data I can stop here. If the requirements are to monitor the data in real time, add or remove data from the database it is possible develop a GUI to do that for local users. If same functionalityies are required for remote users then it's required to develop a web application to do that.

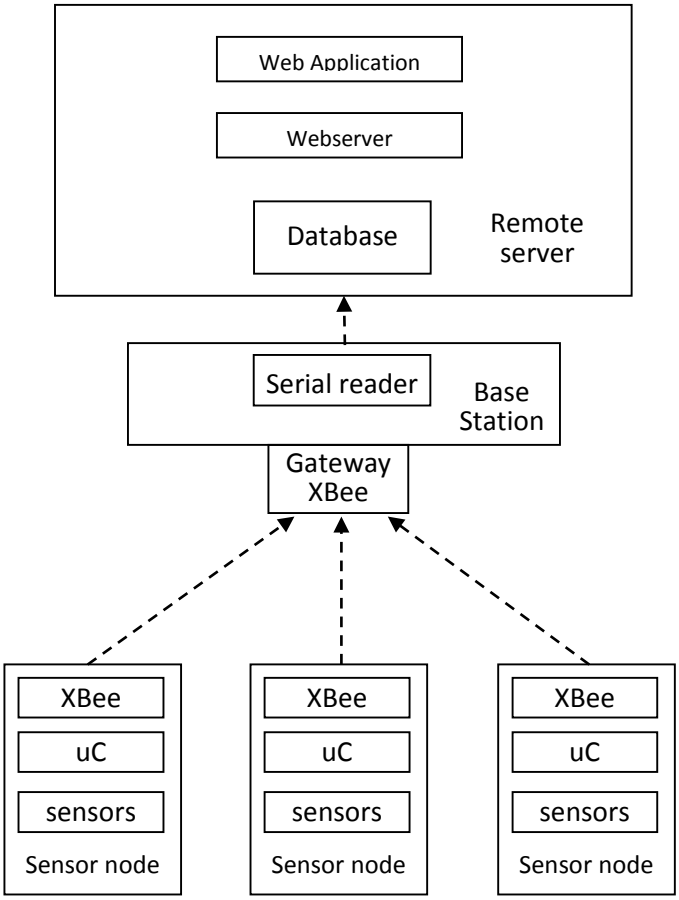


Figure 5.5: Base Station with Separate Webserver

While implementing the multicast option the design gets more complicated because of the introduction of API mode. We can't use the simple serial reader program any more. It is required to write a controller program which can address all the nodes individually and then grab the available serial data. It should parse the received packet and write sensor data into the database. This is close to what I have planned to implement in this design. I went another step further by providing the ability to modify configuration parameter from the client end. I also integrated the gateway and webserver in the same machine to reduce the cost of implementation. As long as a web framework is there more service oriented application can be developed using this system.

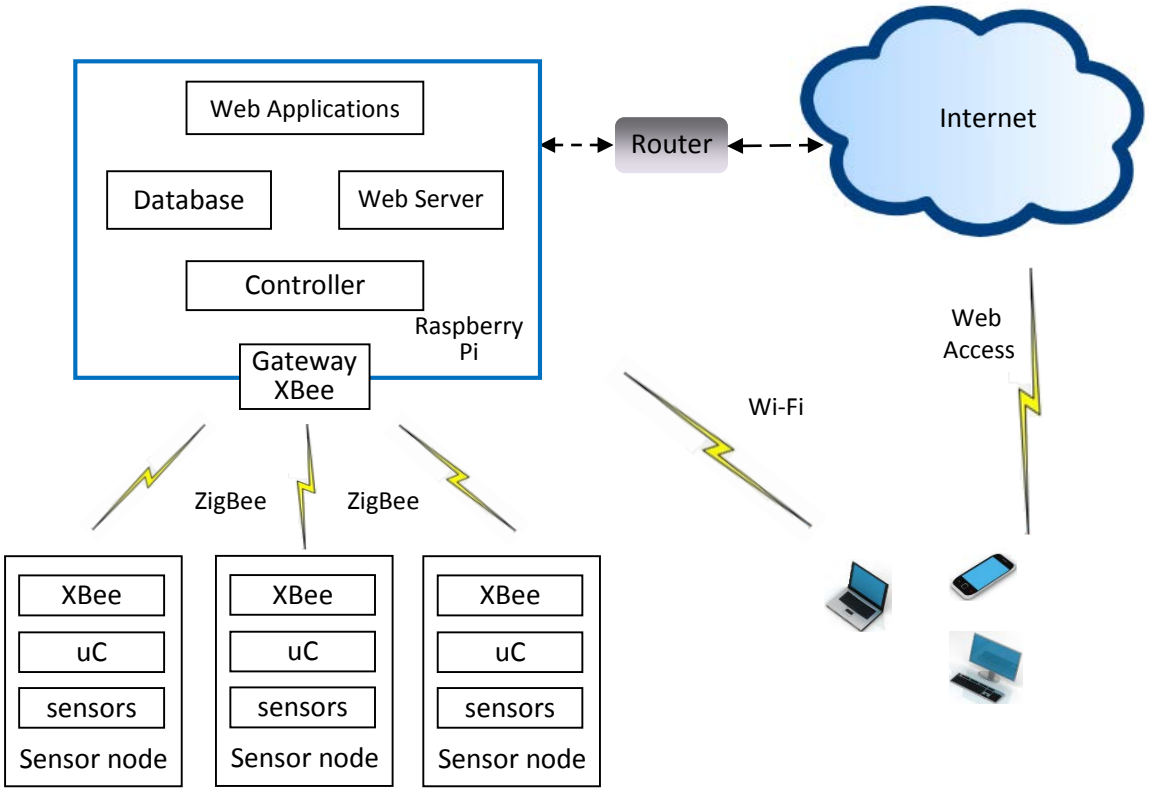


Figure 5.6: System Architecture for the Designed System

CHAPTER 6

SYSTEM DEVELOPMENT

6.1 Introduction

Towards the end of last chapter I have introduced the system architecture of our design. In this chapter I will introduce a much more detailed block diagram of the system and then describe all the components and their design step by step.

6.2 Detailed Block Diagram

The controller was designed for the communication with the sensor nodes and base station. It also had the capability to read and write to a database. It was designed to have two independent processes running simultaneously. One process was used for sending packet to the sensor nodes which was periodically run by the Scheduler. The other process was designed to receive and save the data from various sensor nodes to the Sensor Data Table. Each sensor node was designed with the capability of receiving and processing the packets so that it can send a reply back to the gateway node with appropriate data.

Initially I designed three dynamic html pages to provide service for the remote users. CONFIGURE was responsible for manipulating the CMD Table which had the information how the packet would be formed and where it would be sent. It was also designed to show the status of current configuration. CHART was responsible for plotting sensor data in almost near real time. Using the DATA page I provided access to the MySQL database for the remote users.

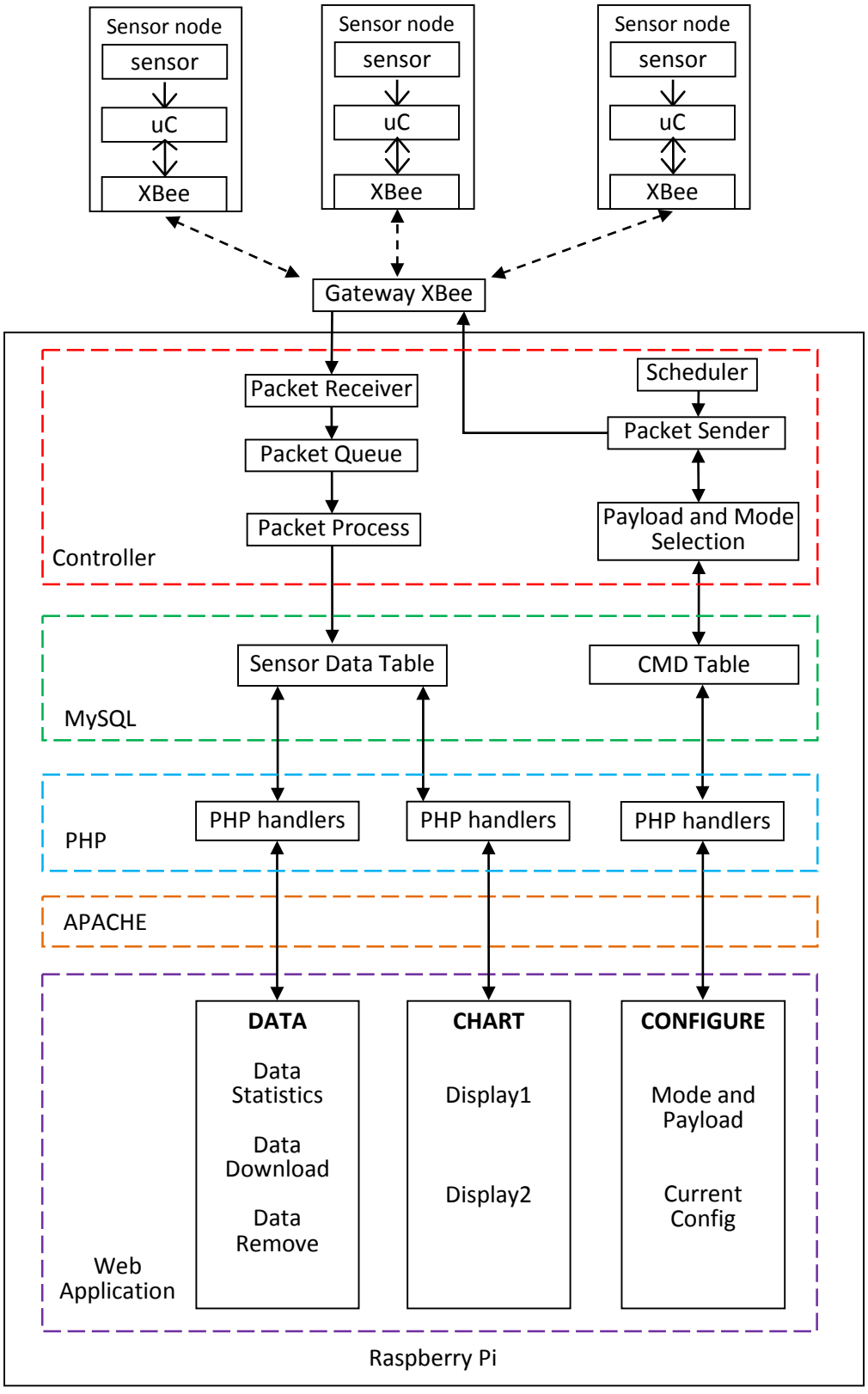


Figure 6.1: Detailed System Block Diagram

6.3 Data Sending Process of Controller

In this section I concentrate on detail design and algorithm of the data sending process of the controller. I implemented an asynchronous mode of communication between the gateway node and the sensor node which means a sensor node would send the data whenever it's asked to do so. Basically from this is the beginning of data collection procedure using the sensor network. I implemented a way to communicate with the sensor nodes in multicast operation. The payload was designed in such way so that which sensor value to return as a feedback by the sensor node can be selected. Following sections will have more discussions on the various components of the data sending process.

6.3.1 Scheduler

The job of Scheduler was to run the function Packet Sender periodically. The Scheduler function has two arguments, one is to select a time period and the other is to select a call back function which is in this case the Packet Sender. I can change the period within the code or there can be an option to change it externally. The period was selected considering quite a few issues. For example, the RHT03 sensor that was used needs at least two seconds for collecting data. If the Packet sender sends data in multicast option it means it has to finish sending frames for all the selected nodes one by one, also the packet receiver have to wait for the individual feedback for each of the selected sensor node. In Packet Queue I have left an intentional pause of 0.5 second. Each MySQL query also takes a small amount of time which is quite unpredictable. All these have to be considered before choosing an appropriate value for the period. In this development I chose 10s as the period which gives ample time to send and receive frames from all the nodes.

6.3.2 CMD Table

I used a MySQL table to contain the configuration information. Configuration information indicates the destination address and the corresponding payload. I designed the table in a very simple method. Each row contains the configuration data regarding that particular node or mode. Column “act” was used to select the mode or nodes. The next two columns were used to create payload for the sensor nodes. Number of rows is scalable based on the number of sensor nodes. Similarly number of columns is scalable based on the number of sensors. 1 or 0 was used to indicate whether that particular mode, node or payload was active or not. It can be accessed and manipulated both from Controller and Client end. Below is the schema of the CMD table:

Table 6.1

Schema for CMD table

CMD		
PRIMARY KEY (ID)	INT	NOT NULL AUTO_INCREMENT
Node	VARCHAR	DEFAULT NULL
Act	INT	DEFAULT NULL
pl1	INT	DEFAULT NULL
pl2	INT	DEFAULT NULL

6.3.3 Payload and Mode Selection

The module MySQL-python was installed before implementing this step. Through this step Controller acquires the configuration data from the CMD Table. It involves a MySQL query which fetches all the row information. Then it constructs two Lists in python; one containing only “act” of the sensor nodes and the other contains combined payload for each sensor node.

Table 6.2 gives an example of a scenario in command table. Below are List1 and List2 based on that table.

List1 = [1,0,0,0] ; List2 = [‘11’,’00’,’00’,’00’]

Table 6.2

CMD Table for Broadcast Mode with Both Payload Active

Node	act	PL1	PL2
bc	1	1	1
sn1	0	0	0
sn2	0	0	0
sn3	0	0	0

Each member of List2 is the result of concatenation of corresponding PL1 and PL2. In the above example payload is only 2 bit wide, as each bit is reserved for a particular sensor. If the number of sensor is five then each payload of List2 would be five bits wide.

6.3.4 Packet Sender

Packet Sender was designed to construct appropriate API Frames and then send the frame data to corresponding sensor node. In section 4.4 the API Frames are discuessed in details. When its is required to transmit data “ZigBee Transmit Request” is used which has an API ID 0x10. If I want to transmit the payload ‘11’ in Broadcast Mode then the Destination Address and Network Address respectively are 0x000000000000FFFF and 0xFFFE. According to the frame structure mentioned in Table 4.2 following packet is constructed.

Byte position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Byte value	0x7E	0x00	0x10	0x10	0x01	0x00	0x00	0x00	0x00	0x00	0x00	0xFF	0xFF	0xFF	0xFE	0x00	0x00	0x31	0x31	0x91

Figure 6.2: ZigBee Transmit Request Packet in Broadcast Mode with Payload ‘11’

Byte 1 corresponds to the start of the Packet. Byte 2-3 is used to declare the length of the packet. Byte 20 is used for checksum. There are 16 bytes between the length and Checksum HEX of which is 0x10. So byte 3 is set to 0x10. Byte 4 is for API ID which is 0x10 for ZigBee Transmit Request. Byte 5 is for Frame ID, we normally set this to 0x01. Byte 6-13 and byte 14-15 are dedicated for 64-bit Destination Address and 16-bit Network Address respectively. Byte 16 is to declare the Broadcast Radius, which is set to 0x00 that means the radius is set to the maximum hop values. Byte 17 gives additional transmission options, normally the value is set to 0x00. Byte 18-19 is used for payload. While transmitting character '11', both 18 and 19 were set to 0x31 since Hexadecimal of character '1' is 0x31.

Now it is time to find out the checksum. To do that I have to calculate the sum of all the bytes between the length and checksum and then deduct the sum from 0xFF.

Cheksum = [0xFF – (0x10 + 0x10 + 0x01 + 0xFF + 0xFF + 0xFF+ 0xFE + 0x31 + 0x31)]

= [0xFF- 0x46E]

= 0xFFFFFC91

I took last two digit as the checksum and set byte 20 to 0x91.

For a multicast operation the destination address is set to corresponding 64-bit Device Address along with corresponding payload. Network Address bytes can be kept 0xFFFE. Length will change if the size of the payload changes. Checksum has to be calculated for each case. Once I transmit this packet using the coordinator XBee module a response is expected back from the destination sensor node which will be another kind of API Frame known as ZigBee Transmit Status. This is discussed in later part of this chapter.

All the above activities and calculations have to be taken care of by the controller for the transmission of every packet. Paul Malmsteen [39] has already designed a module which can be used in python. This module makes the job of packet formation much easier but gives less control on the packet structure. The job of constructing the packet mentioned in Figure 6.2 can be accomplished by the following function:

```
zb.send('tx',dest_addr_long='\x00\x00\x00\x00\x00\x00\xff\xff',dest_addr='\xff\xfe',data= '11')
```

For the first argument of the function I have to use the corresponding API Name for 0x10 which is 'tx'. Rest of the arguments is 64-bit Destination Address, 16-bit Network Address and payload respectively. At the end it came down to only two parameter to choose because for transmitting data I am always going to use 'tx' as the API ID and '\xff\xfe' as the 16-bit Network Address assuming it unknown.

Inside the controller program there is a third list which contains the 64 bit Network Address for Broadcast Mode, sensor node1, sensor node2 and sensor node3 respectively. When act[0] =1 the network starts in Broadcast mode, data is transmitted to all the sensor node and the process ignores the rest of the values in the list "act". Whenever the first value in the list act is 0 it gives the option of having a multicast scheme and the respective payload gets transmitted to the corresponding sensor nodes. For example if act = [0,1,0,1] and pl = ['00','11','01','10'] then 11 and 10 can be used as payload while transmitting packet to sensor node 1 and sensor node 3. No data is transmitted to sensor node 2. Following is the flow chart for the packet sending process. It shows what happens during each run started by the Scheduler.

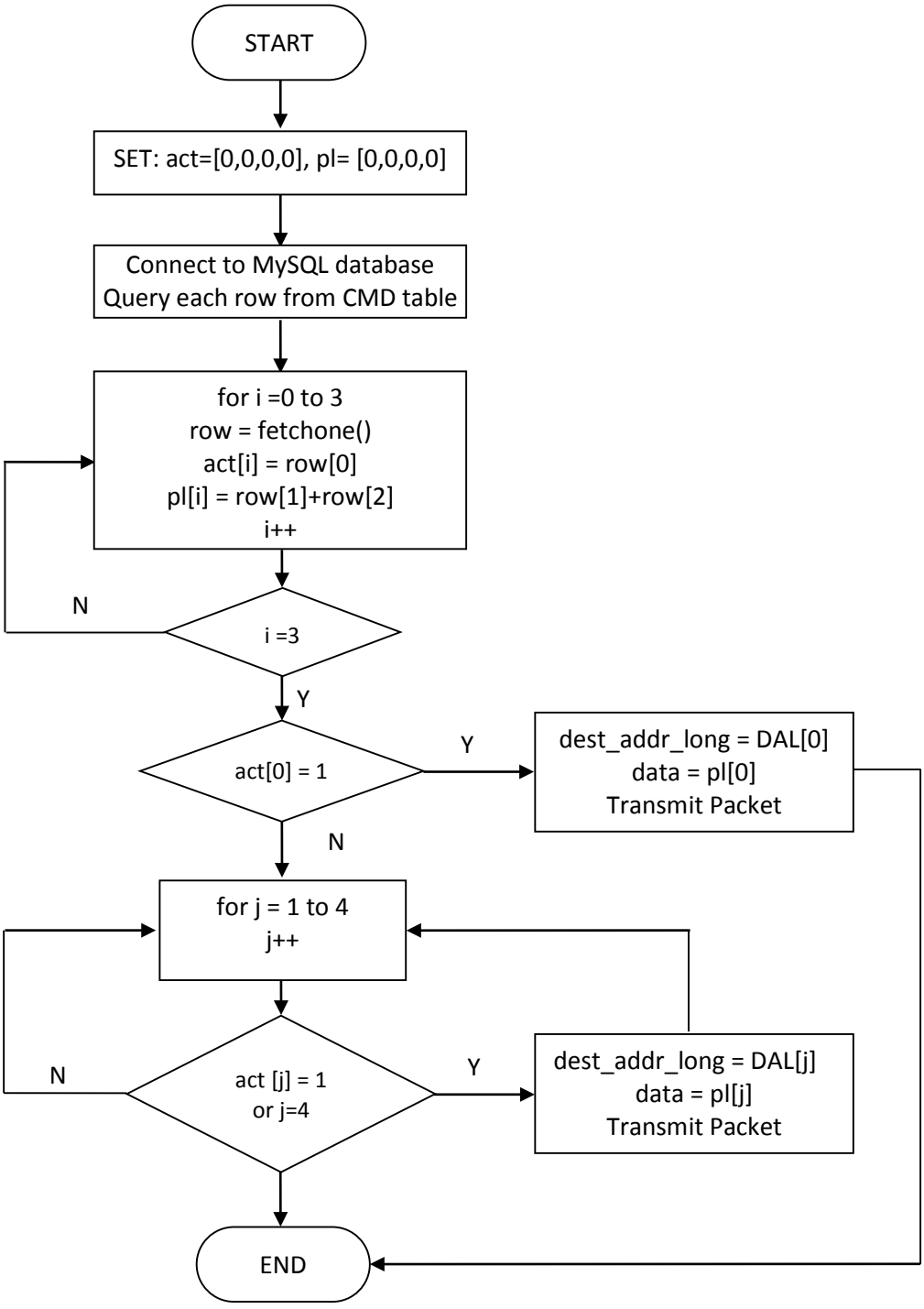


Figure 6.3: Flow Chart for Data Sending Process

6.4 Feedback from the Sensor Node

The job of a Sensor node is to receive packet, decode the payload, prepare and transmit appropriate response back to the controller. For the construction of packet same rule is followed this time the programming environment is different as Arduino IDE is used to do that. For Arduino there is a C++ library [40] which makes it easy to handle the XBee packets. After receiving the transmission from the controller the XBee used in the sensor node does two things, first it sends status frame back to the controller and then it sends UART data to the microcontroller with API ID 0x90.

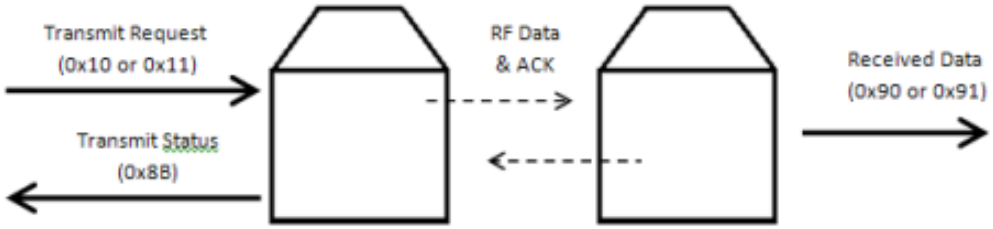


Figure 6.4: Transmitting and Receiving RF Data [38]

Referring to section 4.7.5 it can be seen its possible to get the 64-bit sender address as well as payload from ZigBee Receive Packet with API ID 0x90. XBee library is used to extract the required information from this received data.

Now I will discuss on how the received data is processed and provided a feedback to controller with sensor data. An array named payload is declared having a length equal to 17 at the beginning of the program.

uint8_t payload[17] ={0x68,0x31,0x2c,0,0,0,0,0,0x2c,0x74,0x31,0x2c,0,0,0,0,0}

Byte Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
HEX	0x68	0x31	0x2C	0x30	0x30	0x30	0x30	0x30	0x2C	0x74	0x31	0x2C	0x30	0x30	0x30	0x30	0x30
Char	h	1	,	0	0	0	0	0	,	t	1	,	0	0	0	0	0

Figure 6.5: Payload Design for the Sensor Node.

I filled the bytes 0-2 and 8-11 with the character “h” , ”1” , ”,” and “,” , ”t” , ”1” , ”,” respectively. Rest of the positions was 0. I intentionally hard coded the payload with the sensor tag h1 and t1 which is going to reduce programming load. I used comma in three positions so when the payload was received by the python controller it became easier for it to split the sensor data. Using the DHT library it was quite easy to get the temperature and humidity readings. The problem was those values were float and I had to convert those float values to characters before inserting them inside the payload. Five bytes were allocated each for humidity and temperature data. The unit of Humidity was in percentage and Temperature was degree centigrade. So the value before the decimal sign is expected to be less than 100 and it took two places after decimal, so 5 bytes proved to be enough for each temperature and humidity data. Byte 3 to 7 and 12 to 16 were allocated respectively for humidity and temperature. With the increase of different type of sensors I can increase the size of the payload and insert the respective sensor values accordingly. Below flow chart provides the whole procedure.

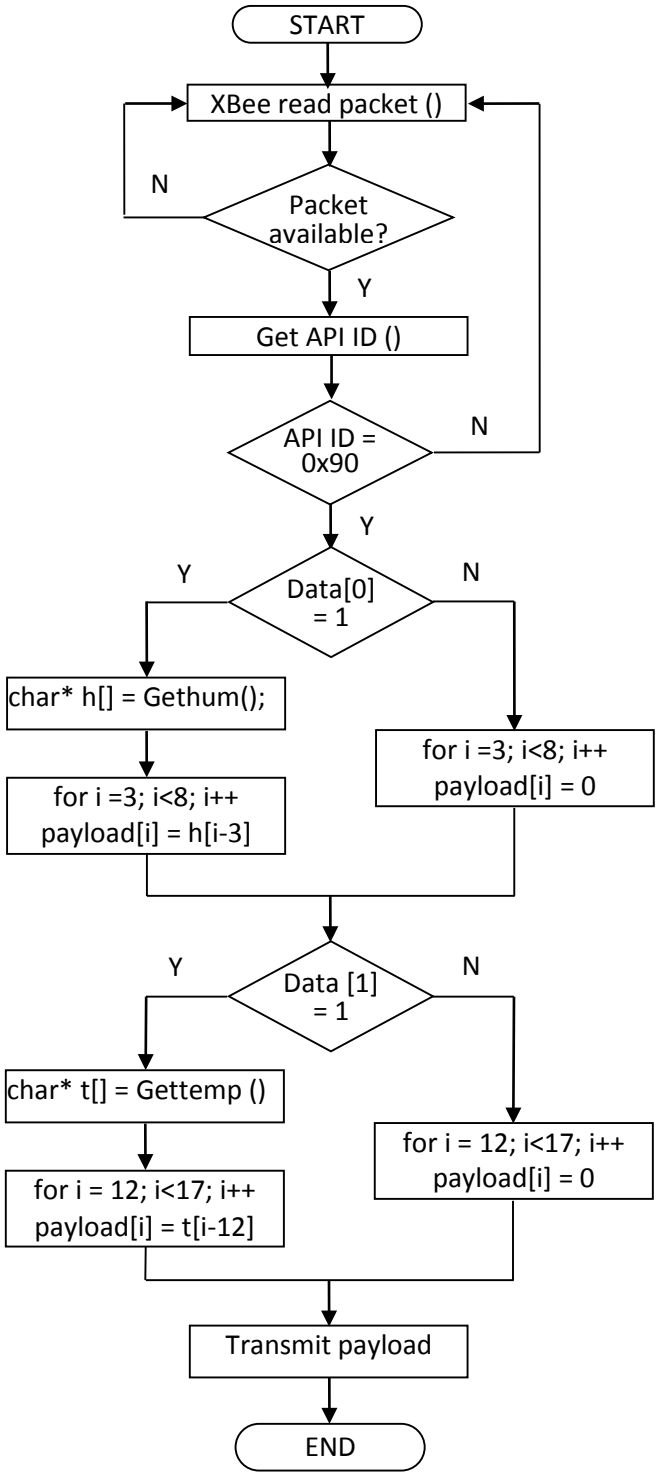


Figure 6.6: Flow Chart for Program in Sensor Node

The read packet function waits for any incoming valid packet. After receiving a packet it checks whether the API ID is 0x90 by using the function getApiID(), if so then it can be assumed its coming from the controller. Then it gets the payload of the ZigBee Receive Packet by using the function getData(). From the previous section it is seen that the size of the payload sent by the controller is 2 byte wide. If first byte is 1 then the program is going to fill the byte 4 to 8 with the humidity value else it fills them with 0. Similarly if second byte is 1 it fills the byte 12 to 16 with temperature data or else those byte gets filled with 0. In this process the payload gets loaded with necessary data. Then it is transmitted with the function ZBTxRequest(). This function has 3 arguments: the destination address in this case its going to be the address of the coordinator, the payload and the size of the payload. All these functions are defined in the library “XBee.h”.

6.5 Data Receiving Process of Controller

The data receiving process is much simpler. The function of Packet Receiver is to save all the available packet through the gateway XBee inside a queue. Then a small routine checks whether the queue size is more than zero, if so the routine calls the function Packet Process. Packet Process strips the received packet and then gets the API ID. The controller is supposed to receive two types of packets, one is ZigBee Transmit Status and the other is ZigBee Receive Packet. ZigBee Transmit Status gives the information whether the data from controller was successfully transmitted to sensor node. ZigBee Receive Packet contains the sensor data that are supposed to receive from each node. All the necessary information is stripped from ZigBee Receive Packet and then inserted into the mysql database. This packet also contains the sender

address which gives the option to identify the sensor node from where it received the packet. The flow chart for the data receiving process is mentioned below.

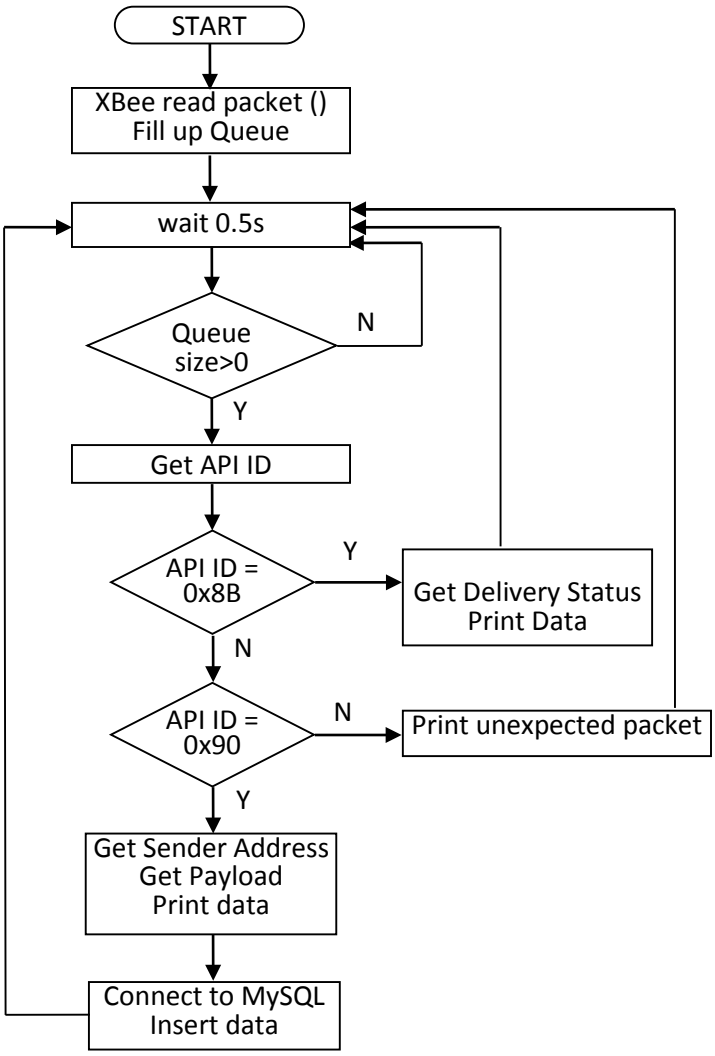


Figure 6.7: Flow Chart for Packet Receiving Process

The sensor data table was designed so that all the necessary information become available. The following table contains the schema for sensor data. Here “sn” is the sensor id, “sal” represents 64-bit long address of the sender, “dt” is the date time, “hl” is the humidity

sensor tag, “hv” is the value of the corresponding sensor. “tl” is the temperture sensor tag and “tv“ is the value of the temperature sensor.

Table 6.3

Schema for Sensor Data Table

SENSOR DATA		
PRIMARY KEY (ID)	INT	NOT NULL AUTO_INCREMENT
Sn	INT	DEFAULT NULL
Sal	VARCHAR	DEFAULT NULL
Dt	DATETIME	DEFAULT NULL
Hl	VARCHAR	DEFAULT NULL
Hv	FLOAT	DEFAULT NULL
Tl	VARCHAR	DEFAULT NULL
Tv	FLOAT	DEFAULT NULL

6.6 XBee Configuration

An import part of the development is to configure the XBee network. The XBee module used with Raspberry was set as the Coordinator. Rest of the XBee modules was set as Router. In the following table I have mentioned the bare minimum parameter that is required to configure the XBee network. The rest of the parmeters were kept intact.

Table 6.4

XBee Configuration for Gateway Node and Sensor Node

Parameter	Gateway Node	Sensor Node
Baud Rate	9600 Kbps	9600 Kbps
Function Subset	ZIGBEE COORDINATOR API	ZIGBEE ROUTER API
PAN ID	4321	4321
Node Identifier	C	R
API Enable	1	2

If PAN ID of the sensor node doesn't match with coordinator then it won't be considered part of the sensor network. I have found out that the python library for XBee doesn't work with AP2, so I set this to 1. On the contrary for all series 2 XBee modules should use AP=2 when it is used as a sensor node otherwise it is not possible to use the C++ XBee library for Arduino. Few parameters like Sleep modes, Power Level can be used to change the characteristic of XBee modules. These are not essential part of the configuration. The coordinator XBee normally finds its own operating channel. So I didn't have to worry about that.

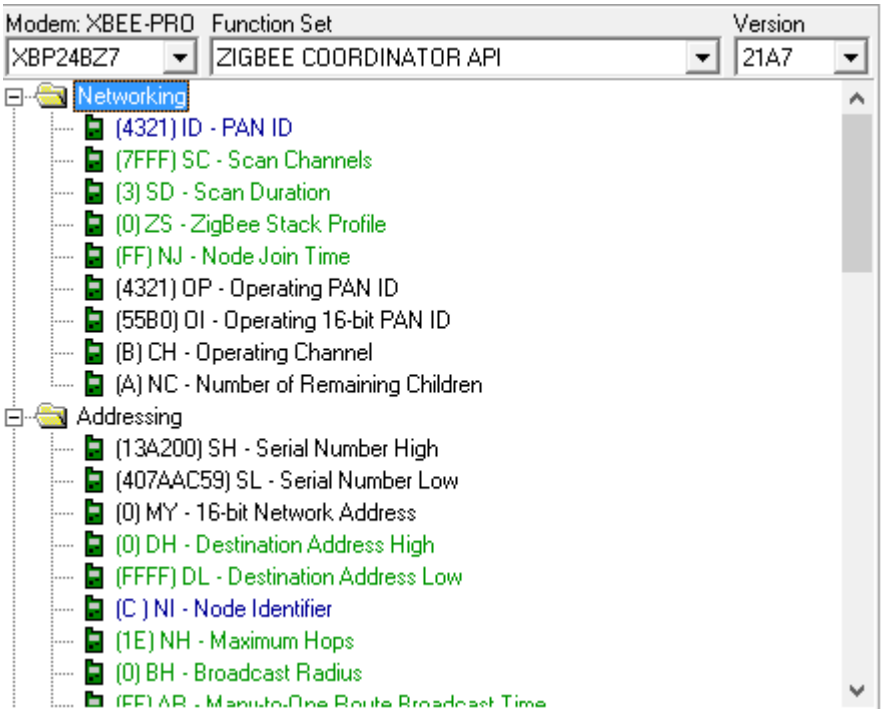


Figure 6.8: Setup Coordinator in XCTU

6.7 Design of HTML Client

The goal of the development was to enable remote access to sensor network and the collected data. I designed a website to serve that purpose. Raspberry Pi is used to host the website. Apache, MySQL, PHP and few other tools were installed. I designed three pages named CONFIGURE, CHART and DATA to accomplish the target. There is option to add more services which can be designed based on requirement.

The page CONFIGURE was designed to manipulate the CMD Table available at the MySQL database. Every time the controller runs it reads the CMD Table. Any changes in the CMD table are translated into the change of destination address or payload. User can directly log in to MySQL from a Linux terminal and manually change the values in the table but the goal was to do it from HTML front end. So, I used an array of checkbox in the front end and a PHP script in backend in such a way so that any time submit button is selected after selecting a checkbox, a “1” will be written to the corresponding cell of the CMD Table. In the same page I introduced another block to give the current status of the CMD Table. I made another PHP script to query the CMD Table which allowed the data to be fetched to HTML front end, write them in a tabular format, so it gives the exact status of the CMD Table.

The page CHART was designed to give a real time plot for both humidity and temperature of a particular sensor node. I created the option to select sensor node, data refresh rate. Flot chart which is a JavaScript module was used to handle the charting function. Using AJAX and JQuery I developed a function which sends an asynchronous request to PHP for data. The corresponding PHP script first finds out for which sensor node the request was sent and then query data from the main Sensor Data table. The query was designed so that it can

fetch latest 10 set of data for the selected sensor node. After fetching the data from MySQL table the PHP script encodes it in serialized JSON format and then transmits data back to the JQuery/Ajax function which plugs the data into the charting tool. So, for each time only the data gets refreshed in the chart instead of refreshing the whole page.

I also wanted to remotely access the data saved inside the MySQL database. The page DATA was designed to serve that purpose. The storage capacity of Raspberry Pi is not unlimited. So, there should be an option to check the condition of the database. To solve that problem I designed a block which could give the number of lines in the main sensor table. Also I created the option to show how many lines are already there for each sensor node with their start and end time. All these information were fetched from the database using queries in PHP script. I designed two more blocks to download and remove data directly from the database. Both for downloading and removing I designed the option to select sensor node, start time and end time. So, it made possible to address a particular sensor within a particular time frame.

CHAPTER 7

EXPERIEMENTS AND RESULTS

7.1 Introduction

In this chapter I discuss how to start and operate the system, show the final look of the HTML pages that have been designed and explain functionality of different blocks. Later on I have shown the collected data in different scenario. I have also done some experiments with this system which is mentioned towards the end of the chapter.

7.2 System Initialization

System initialization is quite simple but there are few crucial steps that have to be taken before running it. It is obvious that all the sensor nodes and the Raspberry Pi itself have to be powered up. Need to make sure the gateway XBee is properly connected to the Raspberry Pi. Once it is connected a port will appear in Raspberry Pi. If it was a windows machine a COM port under Device Manager would appear. When any USB device is attached to Raspberry Pi or any other Linux machine it become visible under the system folder “dev”. To be able to see what is the port name for the gateway XBee the script “ls dev” can be run twice, once before attaching and once after attaching the XBee module. The result is shown in Figure 7.1. A new port named “ttyUSB0” is available after running the second command. If multiple USB devices are connected there can be more of them. In that case the correct one has to be selected. It is important because if the port number is wrong the controller program is not going to run.

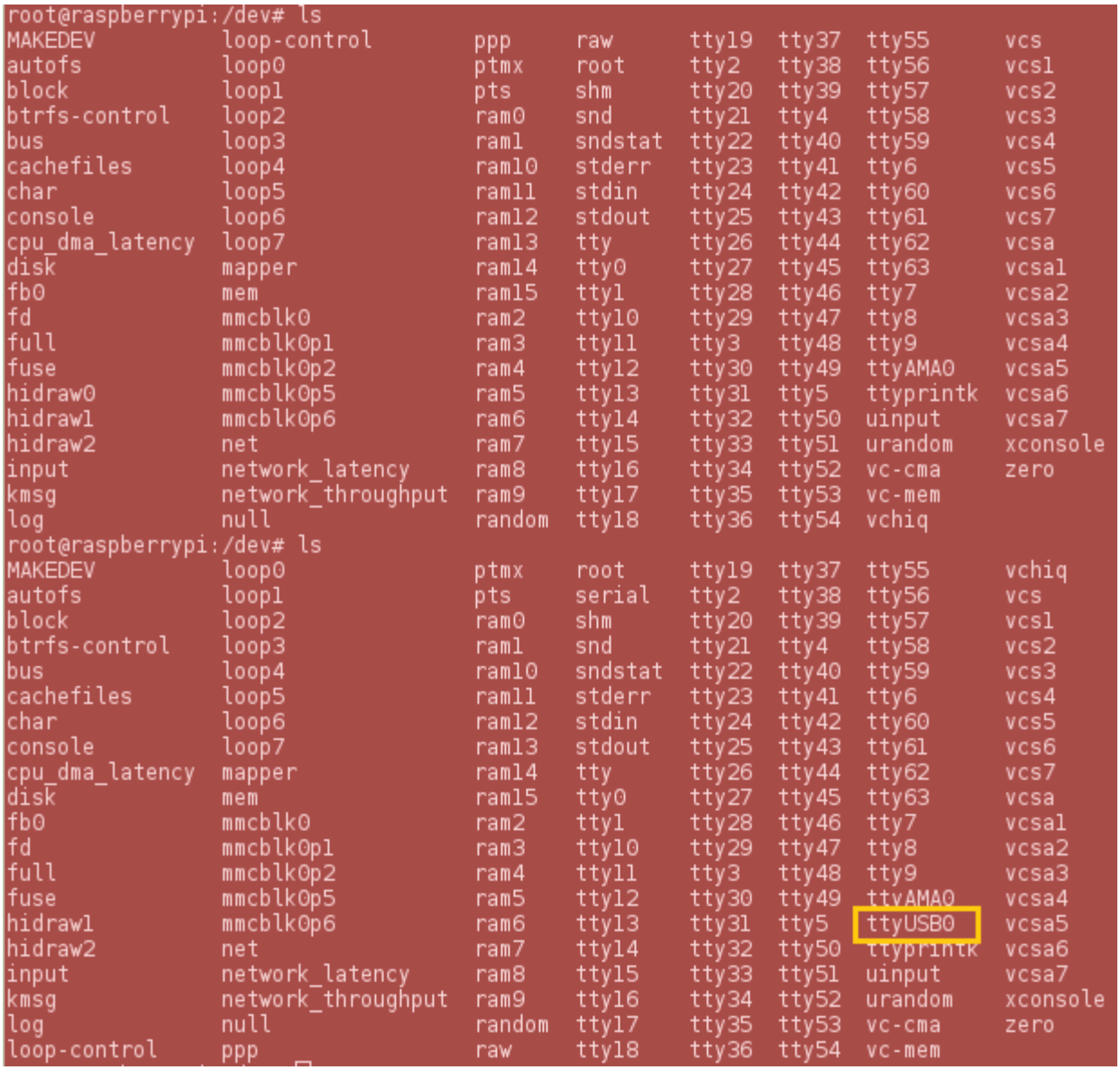


Figure 7.1: Port Opened by the XBee Module

Second thing that also need to be checked is the CMD table, some values should be selected in that table. If all of them are zero then no data will be received though the system is still running. The status of CMD table can be checked by directly logging into MySQL using a terminal. Figure 7.2 shows the query result of the CMD table. The value of act =1 which means Broadcast Mode is selected. It can be seen that pl1 and pl2 value are also 1 which means sensor nodes are going to transmit both humidity and temperature data. If nothing is selected then no

command is transmitted to any of the sensor node and hence data is not received because of the asynchronous nature of communication.

```
mysql> select * from cmtbl;
+----+------+------+------+------+
| ID | node | act  | pl1  | pl2  |
+----+------+------+------+------+
|  1 | bc   |    1 |    1 |    1 |
|  2 | sn1  |    0 |    0 |    0 |
|  3 | sn2  |    0 |    0 |    0 |
|  4 | sn3  |    0 |    0 |    0 |
+----+------+------+------+------+
4 rows in set (0.01 sec)
```

Figure 7.2: Initial Status of CMD Table.

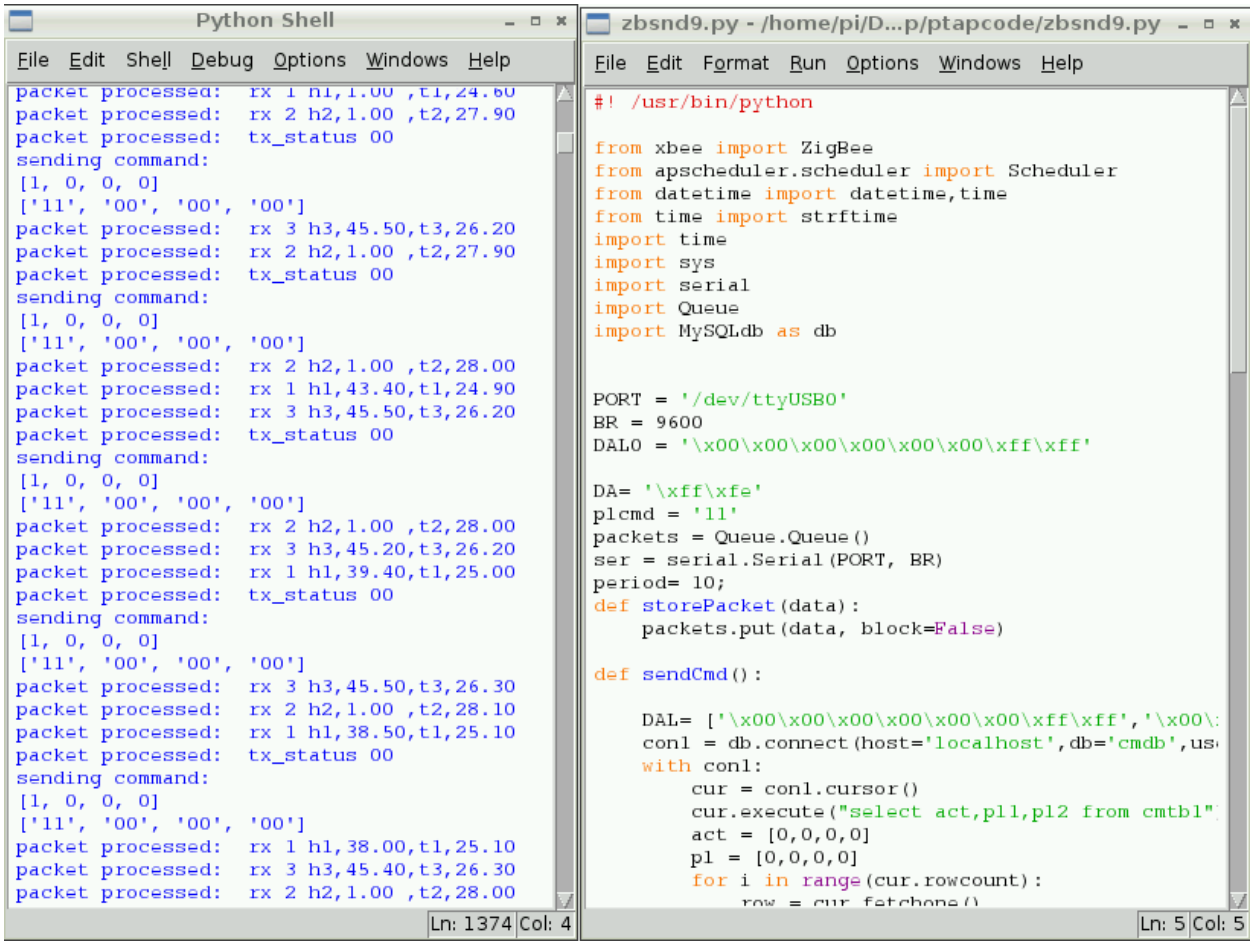


Figure 7.3: Running Controller from Python Shell

To start the actual program a python shell is required using which main controller program can be run. The shell requires a restart to stop the program. The program leaves some

message in the shell during various events. It makes the debugging process easier. Every cycle starts with the message “sending command:” which is then followed by the two lists which define the destination address and payload. Then it writes message for every packet that is received and processed by the controller. For the Received Packet it shows the frame type and received data. For the Transmit Status it writes the frame type and delivery status.

7.3 HTML Client

In this section I show the dynamic pages that have been developed for the system. To start the system configure menu is required. Figure 7.4 depicts the first block which is “MODE AND PAYLOAD SELECTION”. First column in this block decides which sensor node is going to send data, second and third column help to select payload. Individual checkbox can be selected and finally to update the configuration update button needs to be selected. If Broadcast Mode is selected then selection of individual node becomes irrelevant. In case of increase in the number of sensor nodes and sensors, there is going to be an increase of number of rows and columns respectively. Another part of the first block is “PERIOD SELECTION”. It helps to configure the duty cycle of the asynchronous communication between gateway and sensor node. I can write the amount of time and hit update button to change it. The CURRENT STATUS block gives the current mode of communication along with the duty cycle. During the time when the snapshot was taken the network was running in Broadcast mode with the entire payload active and the duty cycle was 6s. Every time update button is selected, it automatically refreshes the page which also updates the data in “CURRENT STATUS” automatically.

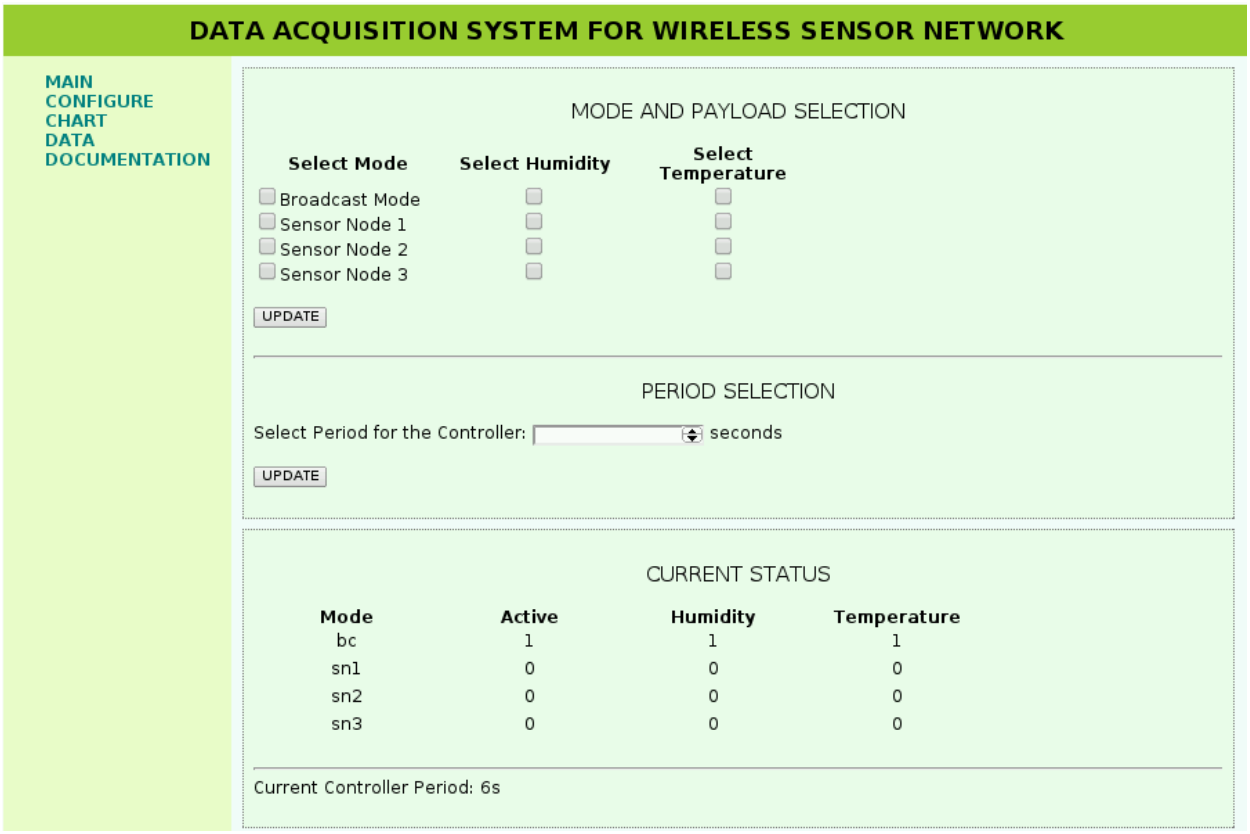


Figure 7.4: HTML Page CONFIGURE

From CHART shown at Figure 7.5 real time plot for the sensor data can be observed. At the top of the page a particular sensor node can be given. After selecting the update chart button the information gets transferred to the concerned PHP scripts which then plot humidity and temperature for that particular node. I also provided the option to show currently selected node. At the bottom there is an option to change Refresh Rate of the chart that defines the period by which JQuery/Ajax functions fetch new data from the database and plug into the chart. Ideally the "Refresh Rate" requires to be greater than or equal to the duty cycle mentioned in the previous page. Float chart is highly customizable which can change the look of the chart for example the axis label, legend, data point value, etc. these options can be added. I planned to use the simplest version.

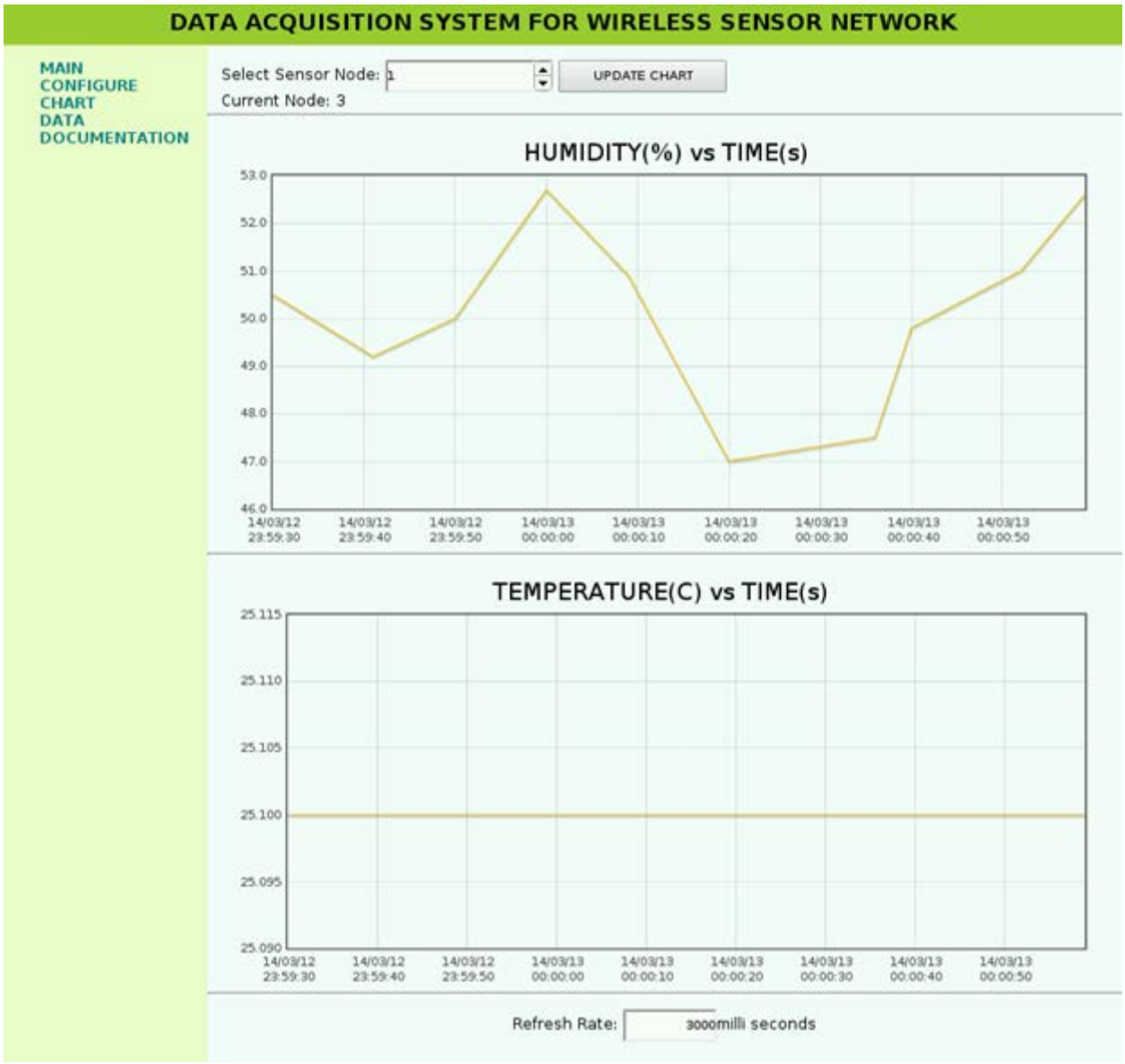


Figure 7.5: HTML Page CHART

The last page is DATA. This page gives remote access to the saved data in sensor data table. At the beginning of the page a lot of information about the stored data is observed. For example, the number of rows currently saved in database. I can find which sensor node data is available along with start date and end date. This makes it easy to download and remove data from the table. The next two sections are used to copy and remove data from the table. There are one click “Download All Data” and “Remove All Data” options which are convenient

sometimes but the later button should be used carefully because it's going to erase all sensor data from the table. If proper back up is not taken then it can prove to be catastrophic. Downloaded data will be saved in client end as CSV format which is convenient for spread sheet analysis. I can also selectively copy and remove data from the table by selecting a sensor node with both the start and end dates, inappropriate selection will return a null csv file.

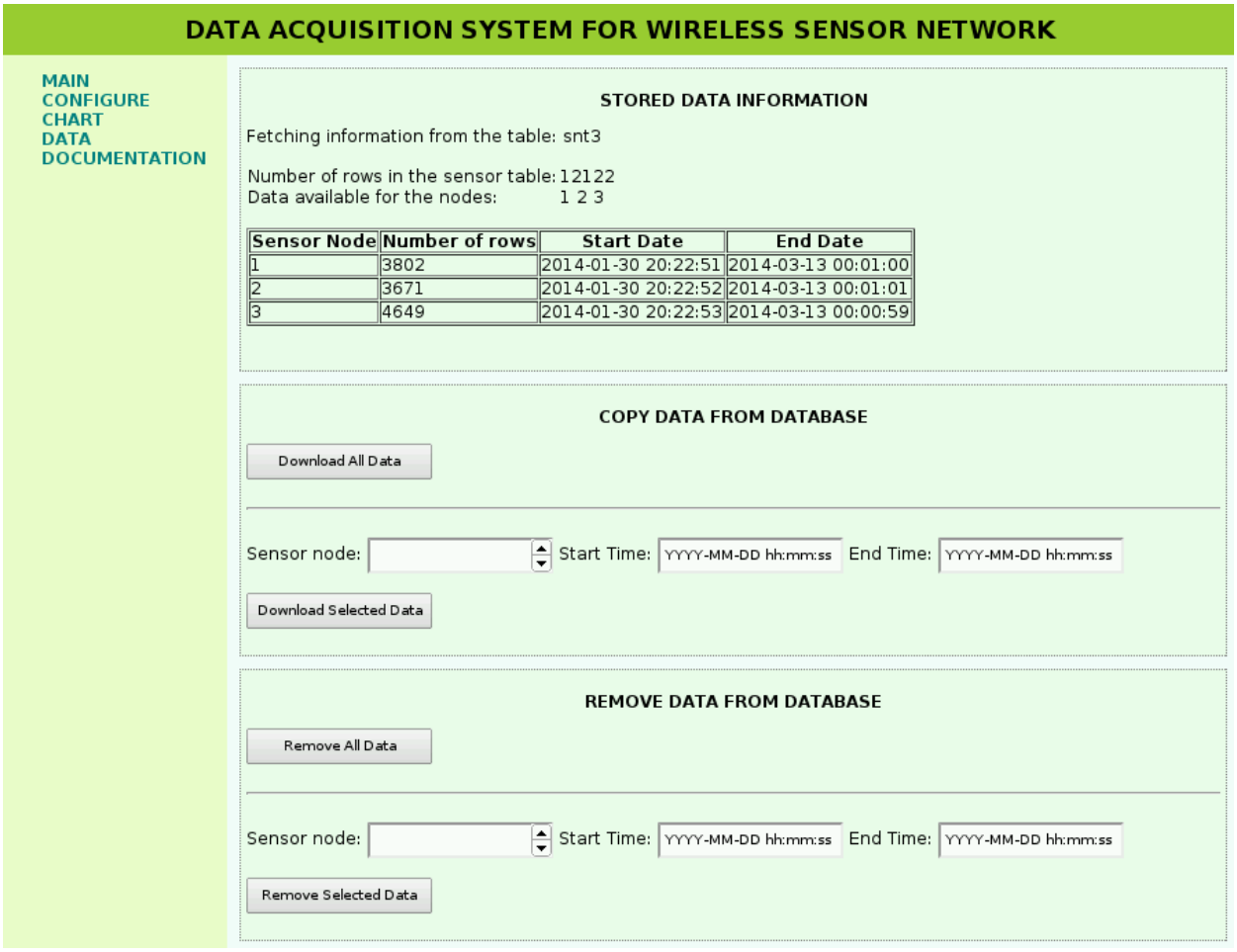


Figure 7.6: HTML Page DATA

"MAIN" was designed to show how to initialize the system and "DOCUMENTATION" was created to contain all the necessary information to build the system.

7.4 Sensor Data Table

I mentioned a lot about the Sensor Data Table. Figure 7.6 gives a glimpse of the table where I did a simple query to show 15 rows of data. Every row has an ID, sensor node tag, 64 bit source address, date time, sensor tag and sensor value. Columns can be added if I plan to use more sensors. Since I have used sensor tag query can be made based on a particular sensor of a wireless node. I can remote login to the server to see the status of the table. If a web based client is not required it is still possible to download data from this table using phpmyadmin.

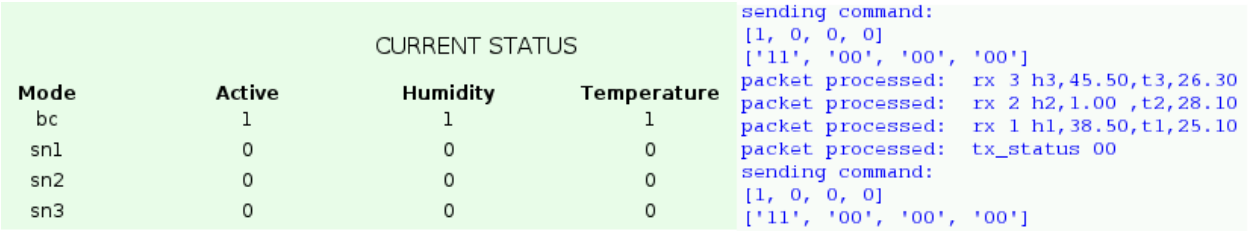
```
mysql> select * from snt3 limit 15;
+----+------+------------------+---------------------+------+-------+------+-------+
| ID | sn   | sal              | dt                  | hl   | hv    | tl   | tv    |
+----+------+------------------+---------------------+------+-------+------+-------+
|  1 |    1 | 0013a200407aac8a | 2014-01-30 20:22:51 | h1   | 51.70 | t1   | 24.20 |
|  2 |    2 | 0013a200407aac88 | 2014-01-30 20:22:52 | h2   |  1.00 | t2   | 24.70 |
|  3 |    3 | 0013a200407aac55 | 2014-01-30 20:22:53 | h3   | 51.70 | t3   | 24.40 |
|  4 |    3 | 0013a200407aac55 | 2014-01-30 20:22:58 | h3   | 51.50 | t3   | 24.40 |
|  5 |    2 | 0013a200407aac88 | 2014-01-30 20:22:59 | h2   |  1.00 | t2   | 24.60 |
|  6 |    1 | 0013a200407aac8a | 2014-01-30 20:23:00 | h1   | 51.40 | t1   | 24.20 |
|  7 |    2 | 0013a200407aac88 | 2014-01-30 20:23:07 | h2   |  1.00 | t2   | 24.60 |
|  8 |    3 | 0013a200407aac55 | 2014-01-30 20:23:08 | h3   | 51.60 | t3   | 24.40 |
|  9 |    1 | 0013a200407aac8a | 2014-01-30 20:23:09 | h1   | 51.50 | t1   | 24.20 |
| 10 |    3 | 0013a200407aac55 | 2014-01-30 20:23:14 | h3   | 51.50 | t3   | 24.40 |
| 11 |    1 | 0013a200407aac8a | 2014-01-30 20:23:16 | h1   | 51.40 | t1   | 24.20 |
| 12 |    2 | 0013a200407aac88 | 2014-01-30 20:23:19 | h2   |  1.00 | t2   | 24.60 |
| 13 |    2 | 0013a200407aac88 | 2014-01-30 20:23:22 | h2   |  1.00 | t2   | 25.10 |
| 14 |    3 | 0013a200407aac55 | 2014-01-30 20:23:24 | h3   | 51.30 | t3   | 24.40 |
| 15 |    1 | 0013a200407aac8a | 2014-01-30 20:23:25 | h1   | 51.50 | t1   | 24.20 |
+----+------+------------------+---------------------+------+-------+------+-------+
15 rows in set (0.01 sec)
```

Figure 7.7: Sensor Data Table

7.5 Experiment with Different Configuration

In this section some experiments are shown with different configuration and observed python shell to see the impact. Current status and python shell messages are assembled side by side to have better understanding. Figure 7.8 shows the message when Broadcast Mode is selected with the entire payload active. In the shell message the configuration data and the

received packets become visible. Three “Received Packet” from sensor nodes are available which is expected, we can see the API type and also the payload. We received the packet “Transmit Status” for which the shell message shows the API type and delivery status.



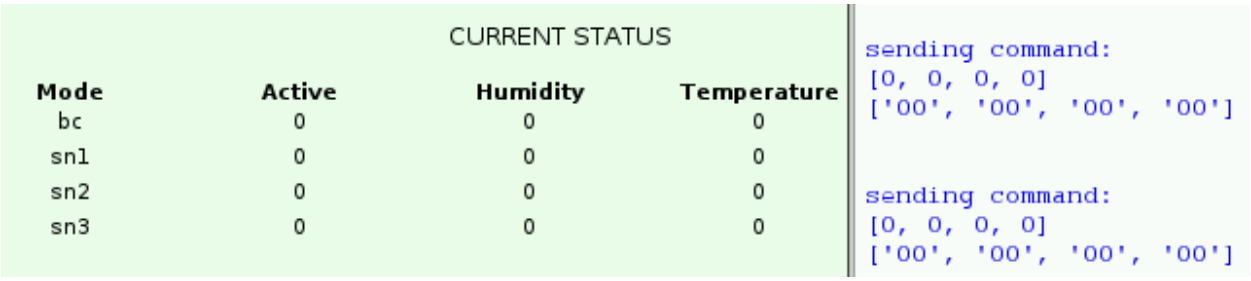
CURRENT STATUS

Mode	Active	Humidity	Temperature
bc	1	1	1
sn1	0	0	0
sn2	0	0	0
sn3	0	0	0

```
sending command:
[1, 0, 0, 0]
['11', '00', '00', '00']
packet processed:  rx 3 h3,45.50,t3,26.30
packet processed:  rx 2 h2,1.00 ,t2,28.10
packet processed:  rx 1 h1,38.50,t1,25.10
packet processed:  tx_status 00
sending command:
[1, 0, 0, 0]
['11', '00', '00', '00']
```

Figure 7.8: Broadcst Mode with all Payload Active

In Figure 7.9 Broadcast Mode also is not configured also any of the sensor nodes is not selected; from shell message it is evident. As a result of this no packets is transmitted between the gateway and sensor node.



CURRENT STATUS

Mode	Active	Humidity	Temperature
bc	0	0	0
sn1	0	0	0
sn2	0	0	0
sn3	0	0	0

```
sending command:
[0, 0, 0, 0]
['00', '00', '00', '00']

sending command:
[0, 0, 0, 0]
['00', '00', '00', '00']
```

Figure 7.9: All Zero Command

In Figure 7.10 I activated all the sensor nodes individually with all payloads active. I received three “Transmit Status” packets as the gateway transmitted individual request to the entire sensor node. I also received the rx packet from individual sensor node. This is a demonstration of multicast mode.

CURRENT STATUS

Mode	Active	Humidity	Temperature
bc	0	0	0
sn1	1	1	1
sn2	1	1	1
sn3	1	1	1

```
sending command:
[0, 1, 1, 1]
['00', '11', '11', '11']
packet processed:  tx_status 00
packet processed:  tx_status 00
packet processed:  tx_status 00
packet processed:  rx 1 h1,16.00,t1,25.10
packet processed:  rx 2 h2,1.00 ,t2,26.00
packet processed:  rx 3 h3,17.80,t3,25.30
```

Figure 7.10: Multicast Mode with All Payload Active.

In Figure 7.11 I intentionally deactivated some of the payload while still in Multicast Mode. In the shell message It is evident that due to this action corresponding data is not sent by the sensor node.

CURRENT STATUS

Mode	Active	Humidity	Temperature
bc	0	0	0
sn1	1	1	0
sn2	1	1	1
sn3	1	0	1

```
sending command:
[0, 1, 1, 1]
['00', '10', '11', '01']
packet processed:  tx_status 00
packet processed:  tx_status 00
packet processed:  rx 1 h1,15.80,t1,
packet processed:  tx_status 00
packet processed:  rx 2 h2,1.00 ,t2,26.10
packet processed:  rx 3 h3,     ,t3,25.20
```

Figure 7.11: Multicast Mode with Some Payload Deactive.

In Figure 7.12 I created a scenario where the network is running in Multicast Mode where all the sensor nodes and payloads are active. I intentionally switched off the sensor node 2. It can be seen “Transmit Requests” were sent for all the sensor nodes and three “Transmit Status” in reply were received from the sensor nodes. One of the Transmit Status has “Delivery Status” 24. By referring to Table 4.5, it means “Address not found” which is expected because sensor node 2 is out of service. From here the importance of “Transmit Status” can be demonstrated.

CURRENT STATUS

Mode	Active	Humidity	Temperature
bc	0	0	0
sn1	1	1	1
sn2	1	1	1
sn3	1	1	1

```
sending command:
[0, 1, 1, 1]
['00', '11', '11', '11']
packet processed:  tx_status 00
packet processed:  rx 1 h1,15.60,t1,25.10
packet processed:  tx_status 00
packet processed:  rx 3 h3,17.60,t3,25.20
packet processed:  tx_status 24
```

Figure 7.12: Multicast Mode with Sensor Node 2 Switched Off.

Since I switched off sensor node 2, it changed the configuration so that no Transmit Request would be sent to that node. Figure 7.13 gives the expected result of doing that. In the python shell there are only two “Transmit Status” as “Transmit Request” is sent to only sensor node 1 and 3.

CURRENT STATUS

Mode	Active	Humidity	Temperature
bc	0	0	0
sn1	1	1	1
sn2	0	0	0
sn3	1	1	1

```
sending command:
[0, 1, 0, 1]
['00', '11', '00', '11']
packet processed:  tx_status 00
packet processed:  tx_status 00
packet processed:  rx 1 h1,15.70,t1,25.10
packet processed:  rx 3 h3,17.80,t3,25.20
```

Figure 7.13: Multicast Mode with Sensor Node 2 Switched Off.

7.6 Sensor Data at Different Location

I collected data using the system at several spots some of the plots are given below.

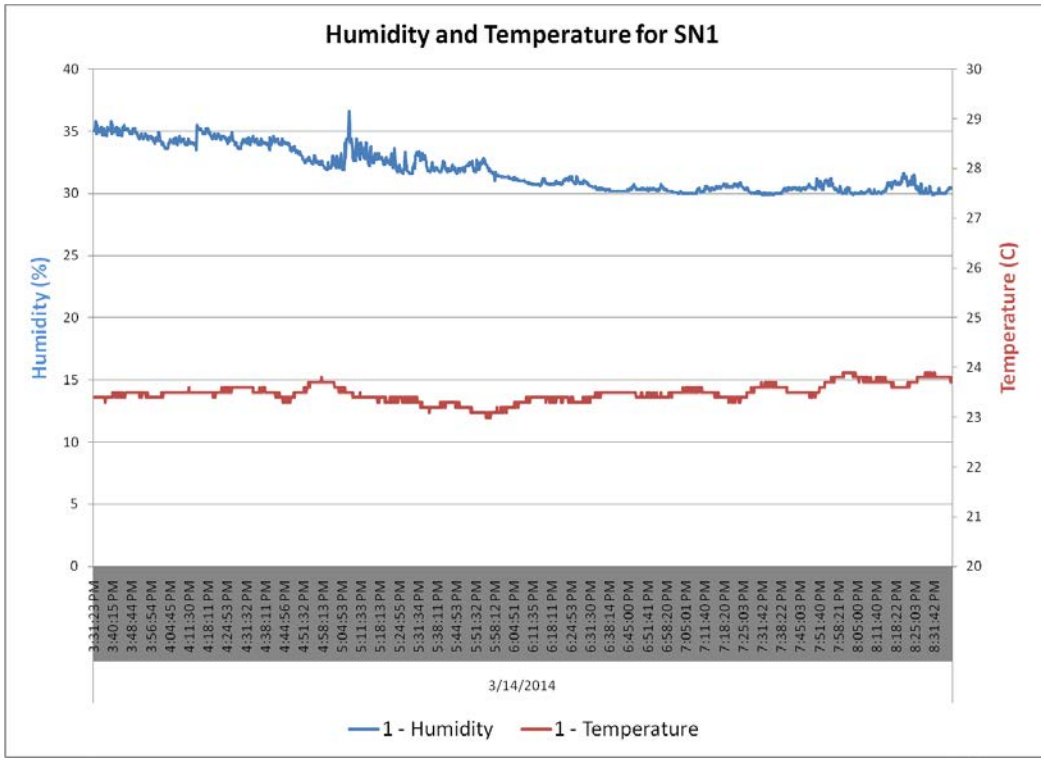


Figure 7.14: Humidity and Temperature at Kitchen of Electrical Department

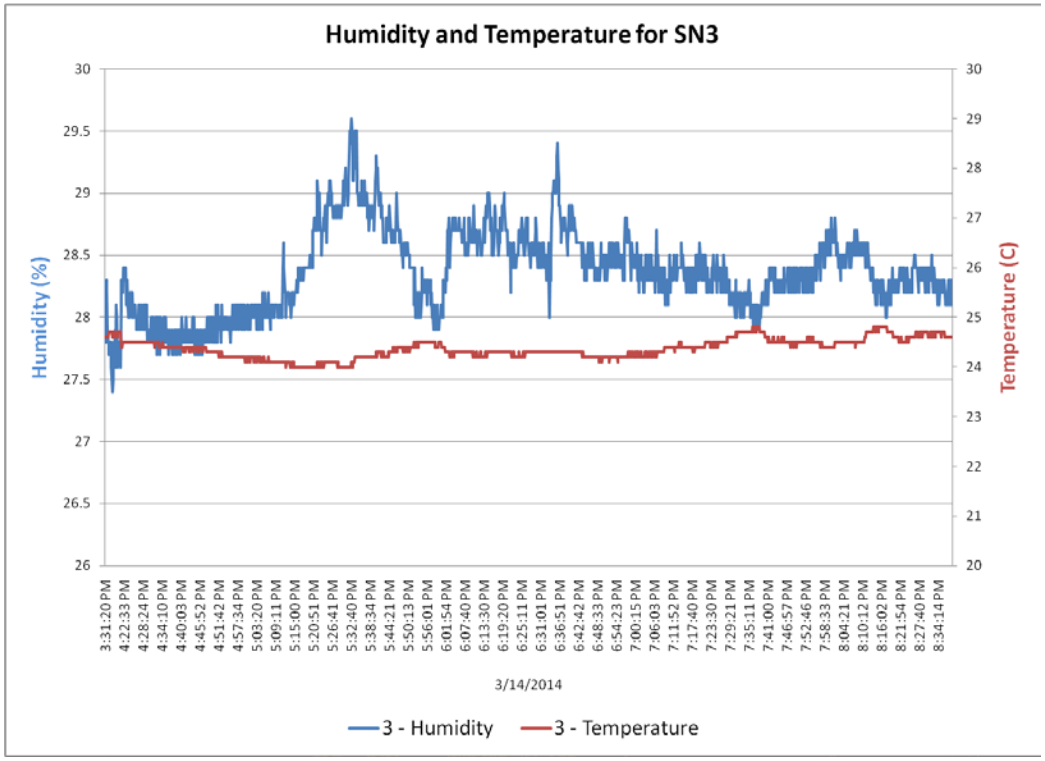


Figure 7.15: Humidity and Temperature at WSSN Lab of Electrical Department

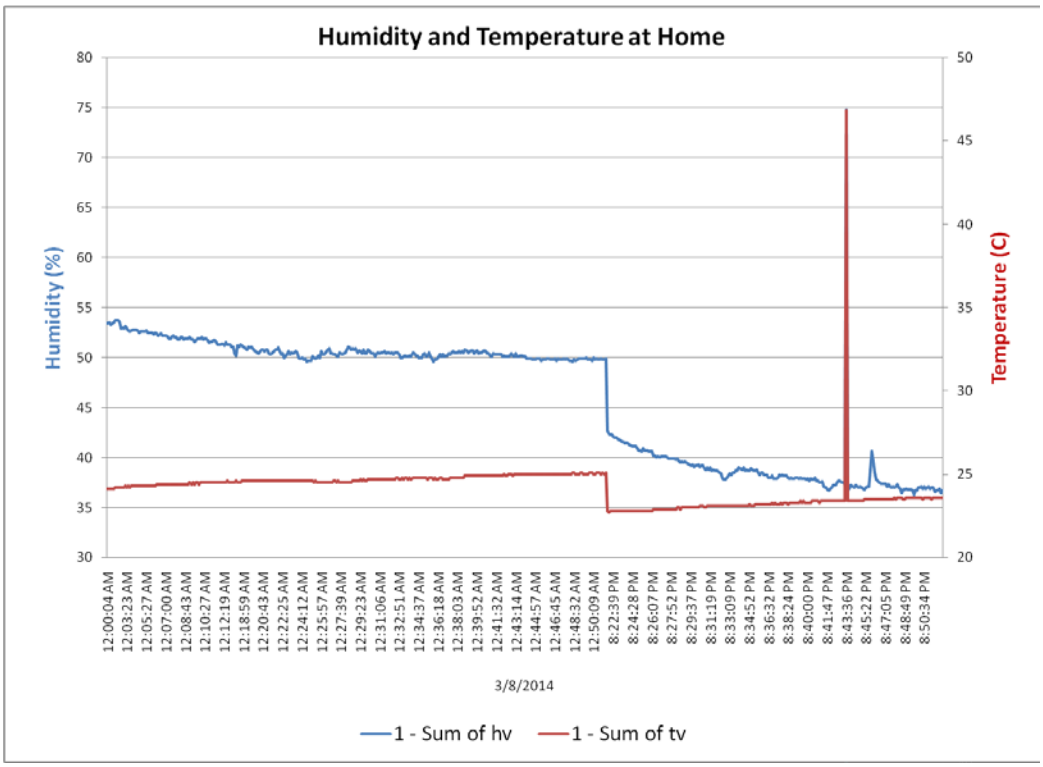


Figure 7.16: Humidity and Temperature at Home

CHAPTER 8

SUMMARY AND CONCLUSION

I have successfully developed a web interface to control and collect data from a wireless sensor network. This system can be considered as a prototype and expand it to serve various applications like Indoor/Outdoor air quality monitoring, home automation system, building management system, environmental monitoring system, etc.

The idea of using a small computer like Raspberry Pi as a webserver itself is amazing. It's cheaper than traditional desktop, laptop or any industry grade server system. Minimum price of a laptop or desktop may be around $300 to $400 whereas the price of Raspberry Pi is only $35. It is also environmental friendly as the power consumption is very low. It can be deployed at an outdoor location for a long term basis. All the software and programming tools used are also open source which keeps the cost of the development restricted to the prices of hardware.

This thesis gave me a unique opportunity to work with low power short range radio technology like ZigBee, open source hardware like Arduino and Raspberry Pi. I also got exposed to web technologies like Apache, MySQL, PHP, JQuery/Ajax, CSS, HTML, etc. It was a great learning curve for breaking the whole development into pieces and step by step implementation of each piece.

I have also successfully used API mode operation for the XBee devices which made it possible to have a fast and smooth multicast operation.

CHAPTER 9

FUTURE WORK

The system requires more testing in terms of reliability before the actual deployment. I haven't seen Raspberry Pi crash a single time during the development and data collection phase. A separate study has to be performed for the long term reliability test under various loads.

More services can be added to generate performance data for XBee network. Data like request fail rate, round trip time for a successful reception from a sensor node, RSSI and available battery level for each sensor node can be generated. This information will give an idea whether the system is running healthy or not.

While developing the system I learned about the web socket concept. By using a web socket a separate port can be opened in the server and have a direct communication link between client and gateway. This provides more flexibility to modify different parameter on controlling application.

The system can be integrated the with a cloud server, so that data can be uploaded to the server time to time. This will ensure the availability of data when the system is out of service.

Few security measures have to be taken before introducing the system to public domain. Seperate user group has to be created so that tasks like modification of configuration or database maintenance strictly belong to administrator.

REFERENCES

[1] Building Wireless Sensor Networks – Robert Faludi

[2] Zigbee Wireless Networking - Drew Gislason

[3] ZigBee Wireless Networks and Transceivers – Shahin Farahani

[4] Getting Started with Raspberry Pi – Matt Richardson and Shawn Wallace

[5] Getting Started with Arduino 2nd edition – Massimo Banzi

[6] ZigBee Alliance. [Online] Available: http://www.zigbee.org

[7] 6LoWPAN. [Online] Available: http://en.wikipedia.org/wiki/6LoWPAN

[8] Apostolos Malatras, Abolghasem (Hamid) Asgari and Timothy Baugé, "Web Enabled Wireless Sensor Networks for Facilities Management", *IEEE Systems Journal, Vol, 2, No.4, December 2008.*

[9] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh, “MoteLab: A Wireless Sensor Network Testbed”, *in Proc Fourth International Symposium Information Processing in Sensor Netoworks, Boise, ID, USA, April 2005, pp 483-488.*

[10] Flávia Coimbra Delicato,Paulo F. Pires,Luci Pirmez,Luiz Fernando Rust da Costa Carmo,"A flexible web service based architecture for wireless sensor networks" *In Proc. 23rd International Conference on Distributed Computing Systems Workshops, May 2003, pp 730-735*

[11] Sajid Hussain, Nick Schofield, and Abdul W. Matin, “Design of a Web-based Application for Wireless Sensor Networks”, *in Proc. 17th International Workshop on Database and Expert Systems Applications, 2006, pp 319-326.*

[12] Xiong Wei, Liu Jian-fu, Zhang Guo-dong, "Applications of web technology in wireless sensor network", *in Proc. 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), 2010, pp 227-230*.

[13] Aishwarya V, Felix Enigo V S, "IP based wireless sensor networks with web interface", *in Proc. International Conference on Recent Trends in Information Technology (ICRTIT), 2011, pp 462-466.*

[14] Rouached M, Baccar S, Abid M, "RESTful Sensor Web Enablement Services for Wireless Sensor Networks", *in Proc. IEEE Eighth World Congress on Services (SERVICES), 2012, pp: 65-72*.

[15] BeagleBone. [Online] Available: http://beagleboard.org/

[16] Raspberry Pi. [Online] Available: http://www.raspberrypi.org/

[17] Arduino. [Online] Available: http://www.arduino.cc/

[18] Arduino XBee Shield. [Online] Available: http://www.cooking-hacks.com/shop/arduino/shields/communication-shield-xb-bt-rfid

[19] XBee module: [Online] Available: http://www.digi.com/

[20] XBee adapter kit: [Online] Available: https://www.adafruit.com/products/126

[21] RHT03: [Online] Available: https://www.sparkfun.com/products/10167

[22] Raspberry Pi quick starg guide: [Online] Available: http://www.raspberrypi.org/wp-content/uploads/2012/04/quick-start-guide-v2_1.pdf

[23] Raspberry Pi OS installation: [Online] Available: http://learn.adafruit.com/setting-up-a-raspberry-pi-with-noobs

[24] Raspberry Pi operating system review. [Online] Available: http://www.techradar.com/us/news/software/operating-systems/raspberry-pi-operating- systems-5-reviewed-and-rated-1147941

[25] Raspberry Pi distro test. [Online] Available: http://www.linuxuser.co.uk/reviews/distro-super-test-pi-edition

[26] Comparison of webserver. [Online] Available: http://www.whisperdale.net/11-nginx-vs-cherokee-vs-apache-vs-lighttpd.html

[27] MySQL. [Online] Available: http://www.mysql.com/

[28] PHP. [Online] Available: http://www.php.net/

[29] Phpmyadmin. [Online] Available: http://www.phpmyadmin.net/home_page/index.php

[30] Flot chart. [Online] Available: http://www.flotcharts.org/

[31] Jquery. [Online] Available: http://jquery.com/

[32] Ajax. [Online] Available: https://api.jquery.com/jQuery.ajax/

[33] Image source. [Online] Available: http://www.pcmag.com/article2/0,2817,2407058,00.asp

[34] Image source. [Online] Available: http://www.bit-tech.net/modding/2013/01/22/bit-tech-raspberry-pi-case-competition/1

[35] Image source. [Online] Available:http://www.bricogeek.com/shop/14-arduino-xbee-shield.html

[36] Image source. [Online] Available: http://www.jerome-bernard.com/blog/2013/01/22/arduino-with-xbee-and-rgb-led-strip-talking-to-beaglebone-with-xbee/

[37] Image source. [Online] Available: http://robosavvy.com/store/product_info.php /cPath/ 1082/ products_id/2784

[38] ZigBee productuct manual . [Online] Available: http://ftp1.digi.com/support/documentation/90000976_P.pdf

[39] XBee api for python. [Online] Available: https://code.google.com/p/python-xbee/

[40] XBee api for python. [Online] Available: https://code.google.com/p/xbee-arduino/