UNIVERSITI
TEKNOLOGI
PETRONAS

Final Year Project

Final Report

Development of a Robust Wireless Sensor Mesh and Multi-hop Network

Supervisor : Dr. Micheal Drieberg

Ahmad Muhaimin bin Mohd Taib

Electrical and Electronic Engineering

11872

ABSTRACT

Wireless networking has evolved rapidly since the first wireless device was invented. Throughout those years, researchers and engineers are struggling to apply the knowledge of wireless networking in useful ways in real life. Wireless Sensor Network (WSN) has been used in many applications, from habitat surveying to critical monitoring. Reliability of the WSN plays a major role in deciding whether it should be used or not in critical applications instead of using traditional wireless technology or wired networking. This project is solely a research and development of routing algorithm for WSN by using an existing source and straight away finding its weak point in order to apply further improvisation. The existing routing algorithms used are the XMESH and Ad-Hoc On-Demand Vector Routing (AODV).

ACKNOWLEDGEMENTS

My very first acknowledgements go to Universiti Teknologi Petronas, Final Year Project coordinator and anyone involved for allowing me to be able to conduct this great project that gave me a new experience about Wireless Communication.

I would like to express my very great appreciation to all the persons in Electrical and Electronics Engineering Department for their valuable and constructive suggestions during the planning and development of this project.

I would like to express my deep gratitude to Dr. Micheal Drieberg, my Final Year Project's supervisors, for his patient guidance, enthusiastic encouragement and useful critiques of this research work. His willingness to spend his valuable time so generously has been very much appreciated. This project will not be able to be completed without his assistance by my side.

I would also like to extend my thanks to my family for support in terms of encouragement and finance thus allowing me to finish this project and research. I am also indebted with my friends in UTP for helping me conducting certain experiment that is hard to be done alone.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Background

Wireless sensor network (WSN) gives additional approach to wireless networking. It generally means a network that comprises bunch of nodes that work together to achieve one objective that is to provide the base station its sensor data as well as maintaining the network. The nodes have its own sensor to detect physical or environmental changes, for example coordinates, temperature, vibration, humidity, and accelerometer [1].

Wireless mesh network (WMN) is a network that consists of two or more devices that can organize themselves by automatically create its own network and finding an alternative route when there is a node failure which is called self-organized and self-healing system. A node is merely an electronic device in a network that can receive, transmit or as intermediate path of data communication [2]. A wireless mesh network may consist of mesh router and mesh network, the mesh router contain the gateway and other routing information in supporting the mesh network, whereas the mesh client is merely an endpoint node [3].

“Robust Wireless Sensor Mesh” in this project title means the combination of wireless mesh network (WMN) and wireless sensor network (WSN) to produce a networking system that can transmit critical data regardless of the situation to the base station. It also need to be a low-powered device and cost effective system where the system does not need major care or service, it can organize and heal itself if there are minor device failure.

1.2 Problem Statement

Sensors are needed in order to measure environmental changes and physical changes of surroundings or specific substances. Distributing sensor devices in a wide area will be challenging for some cases involving narrow space, tough geographical texture and wide area of sensor implementation. Standard wireless networking such as Wireless Local Area Network (WLAN) or just normal Radio Frequency may not be appropriate to be applied in some cases such as natural disaster monitoring situation. WLAN needs a centralized router to function properly and consumes a lot of power. The old wireless technology cannot provide wide coverage except by placing the router/gateway everywhere that will cost a lot to manage and install.

Although researchers and manufacturers nowadays did a good job in developing Wireless Sensor Network to be applied in various areas including the above cases, still there are so many problems involved with this new technology due to its reserved and limited resource and power. Reliability or may be called "robust" of the system is toughly being criticise, packet data dropping, data aggregation and fault tolerance are the main concern [4]. Previous study has reported that in some wireless sensor network designed, the increase of hop and distance for end-to-end communication from a node to the base station cause a rapid increase in packet drop [5]. High latency also reduces the functionality of the system as data is needed almost in real time for some applications. The limited power source and increase in the number of sensor nodes will absolutely worsen the above problems in field application compared to theory.

1.3 Objectives and scope of study

The main objective of this project is to produce or develop a wireless sensor networking that is “robust”, which means it is reliable and it can work in a rough environment where the system will organize itself for any situation without the need of human interaction. It needs to be able to adapt to a new environment almost instantly where a new sensor node may be installed without reconfiguration. A long lifetime is a must as the sensor may be placed on a variety of geographical texture, so power source is precious.

Scope of studies will involve wide scope of communication system such as network protocol, networking standard, signal processing, device interfacing and network layer. Study of wireless communication is a must to be able to understand how the system works and implementing it. In deploying a working Wireless Sensor Network (WSN), knowledge of C language programming is needed to be able to program a microprocessor or microcontroller which may be used in the project design. In improving the WSN, the routing protocol used, XMesh, need to be fully understood to alter the existing protocol in making the system better. Experimenting with different parameter of the devices such as transmission power, data interval, duty cycle and radio frequency will require the programming skill and basic network system understanding.

CHAPTER 2

LITERATURE REVIEW AND THEORY

Distributed sensors network is used everywhere from monitoring the natural disaster occurrence to just measuring temperature in multiple room of a building. Take temperature measurement for example, in a secured laboratory building where some place need to be in a certain temperature, distributed sensor network plays an important role to monitor all the rooms temperature. Wireless sensor network (WSN) can be constructed in many ways, most of the times they use mesh networking topology for it to be easily managed and cost effective. For natural disaster, landslide can be predicted by measuring the vibration and movement of soil, a wired sensor system will not be effective for long time monitoring as the connection cable may be accidentally broken. WSN will solve the problems by introducing wireless transmission for sensors to communicate without sacrificing much the size, power and cost requirement. Mesh networking is a type of network topologies where the wireless devices in the system do not just communicate with a central backbone device, they can communicate with each other and also forwarding data towards one another to achieve a single large network, also called mesh cloud. Using mesh topology on the WSN system will create a lot of advantages over the old system.

2.1 Wireless Sensor Network (WSN)

Wireless Sensor Network (WSN) is basically a bunch of nodes (motes) that are distributed over a wide area which they can pick up the surrounding physical changes via their equipped sensors and communicate with each other in sending the sensor's data to a base station or monitoring station. They use wireless technology as the communication medium to ease the transmitting of data without the need of connection cables. The nodes are limited in term of resources including power, transmission, memory and size. The wide diversity of application in real life such as military, home, industry, environmental and also health make it attractive to be researched and improved **[**6**]**.

2.1.1 Node’s Components

Nodes can be either identical or different, it depends on the system topology that is being used. A node may consist the following:

1. Sensor device – an electrical circuit that can detect analog measurement of real world environmental and physical changes, then converting it to digital signal for further processing [**7**].
2. Transceiver – wireless transmitter combined with receiver, its main job is receiving signal from other nodes and hand it over to CPU to be processed. Data from CPU is taken and transmitted to other nodes.
3. Central Processing Unit (CPU) – usually a low-powered microcontroller or microprocessor board. It is commonly included with its own constraint memory and cache onboard for program storage and Random Access Memory (RAM) to be used by the processor. This unit serve as the main part in processing the data acquired from the transceiver and sensor board for further task.
4. Gateway board – An interface board that allow the CPU to interact with other kind of devices, supposedly a monitoring computer or an internet gateway to send data to the server.
5. Power storage – mostly batteries for powering up the Sensor board, Transceiver and CPU. It is ranged from 2 to 6 volts and all kind of battery including dry-cell, alkaline, rechargeable cell.

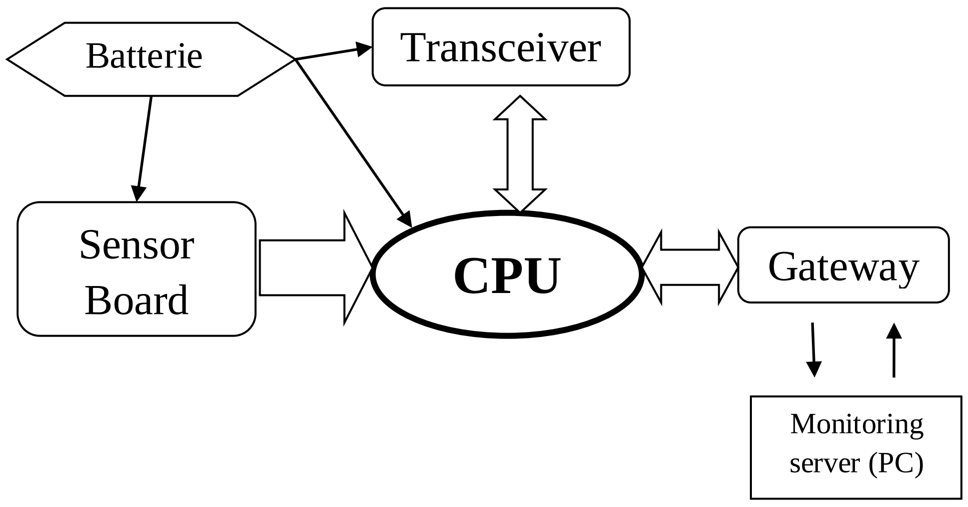


Figure 1 Simple node's component illustration

For full-mesh topologies, all the nodes are identical except for the base station that needs a gateway to interact with human monitoring devices or server. Figure 1 shows the basic illustration of component inside a node with additional gateway. Batteries are powering the three main components of the nodes, in some design, the gateway also need to be powered too. The sensor board may contain one or multiple type of sensor(s), for example, Global Positioning System (GPS), humidity, temperature, accelerometer, movement, ambient light, vibration and so on depending on the application requirement.

2.1.2 Wireless Sensor Network (WSN) Topologies

In designing or building the Wireless Sensor Network (WSN), there are three main topologies that can be used.

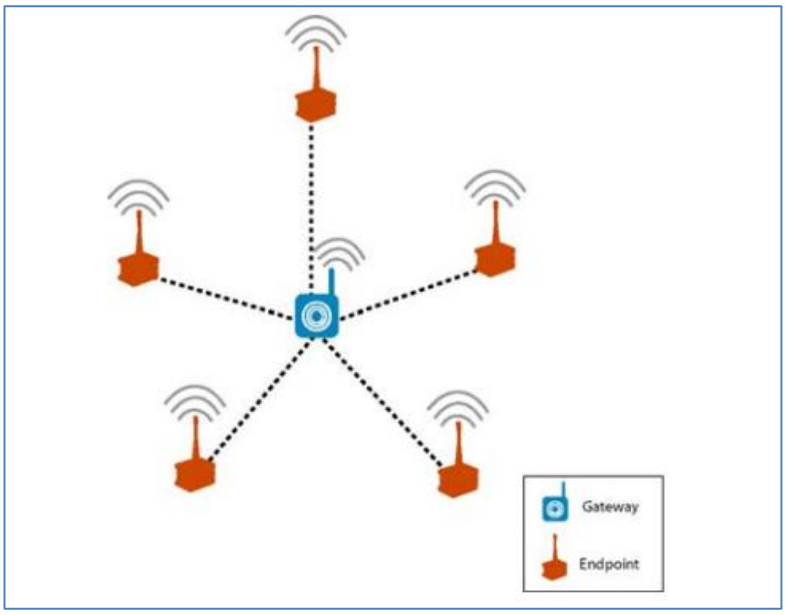


Figure 2 Star topology (Source: Ref [8])

1. <u>Star network</u>

As shown in Figure 2, a centralized node is located usually at the centre of a network surrounded by endpoint node as the sensor node. All the sensor nodes are only able to transmit its sensor's data to the central node without being able to perform as intermediate node for data relaying. So generally the star topology is a single-hop network that can be operated within 30-100 meters from the centralized node (gateway) **[**8**]**. The main advantage of this type of network is that the endpoint/sensor nodes require least amount of power where the main gateway required being line-powered. The major disadvantage is that the coverage is not that good compared to other topologies with same amount of power and cost used.

2. Hybrid-star network

This type of network comprises multiple centralized nodes/gateways that are connected to each other. Behind each gateway, there are sensor nodes that that will sense the change of physical and environment. These sensor nodes will be able to communicate not only with its existing parent gateway, it can also pair up with other gateways in case of any gateway failure as in Figure 3.

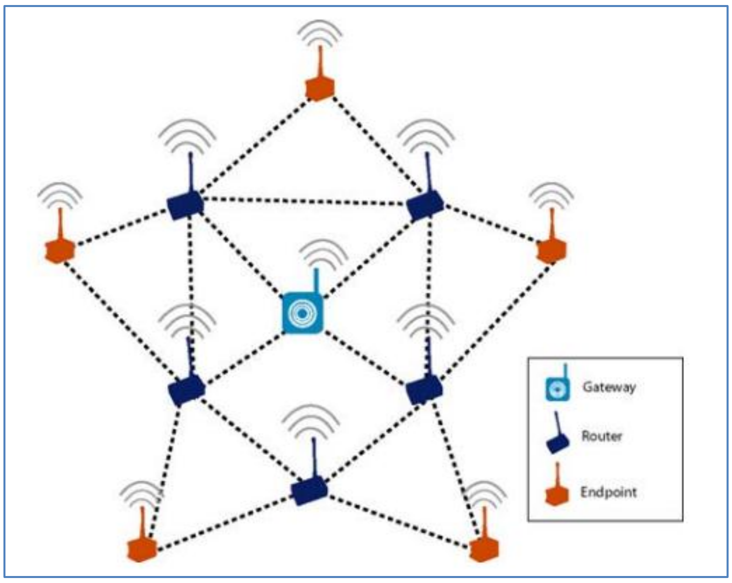


Figure 3 Hybrid-star/mesh topology (Source: Ref [8])

If the design can support multiple line-powered routers among the distribution of devices, then this type of network is suitable as it is a simple hierarchical network. The endpoint nodes just do a simple task of gathering the sensor data and transmitting the data to the router, where the router then perform its communication task. The advantages of this network is the extended lifetime of the sensor nodes which can be battery-powered despite the router need to be active all the time to route data **[**9**]**.

3. <u>Full mesh network</u>

It is a network where all nodes are identical, they can acts as sensor nodes, gateway and also as an intermediate node.

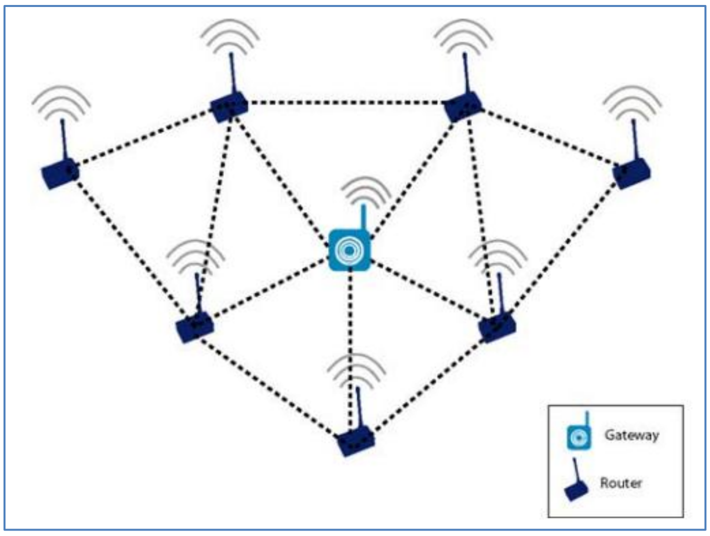


Figure 4 Full-mesh topology (Source: Ref [8]**)**

The devices communicate with one base gateway by multi-hop to one another, this vastly increase the coverage availability, theoretically infinite coverage. It is a self forming and self healing because of the multiple routes available in case one is broken **[**9**]**. When one node is faulted, the network will reconfigure itself to find another route/path for the data to be transmitted to the gateway or another node. This topology commonly needs one base station, in Figure 4, the Gateway acts as the base station for server monitoring.

2.1.3 WSN Applications

WSN has wide areas of application in real life situation, some of it may be critical and some may be not. These are some example of applications that are already being implemented.

1. Chemical monitoring – it has been deployed in some cities to measure the concentration of poisonous substances that may endanger the citizen. The WSN will be an advantage as it will easily covered wide areas in a city and making them easily tested by moving the nodes to different areas **[**10**]**.
2. Greenhouse monitoring – the environment parameter inside a greenhouse need to be carefully monitored such as the suitable temperature and humidity
3. Landslide warning – the detection of vibration and movement at the possible occurrence of landslide can give the surrounding resident an early warning before a landslide can occur.
4. Patient health monitor – the use of WSN will make it easy for medical institution to apply the wireless centralised monitoring system of critical patients.
5. Restricted area monitoring – military or government can use the WSN technology to monitor intrusion of personnel into a restricted zone or areas by detecting movement of people near the fence.
6. Habitat survey – monitoring of an endanger species in its habitat can easily be done by placing nodes with the required sensors. The less maintenance, high latency, long lifetime system may be used for this application.
7. Infrastructure monitoring and analysing – massive infrastructure such as long bridge, underground tunnel and tall building are able to be constantly monitored wirelessly despite of having personnel to do the visual inspection of the infrastructure.

2.2 Wireless Mesh Network

Wireless mesh network (WMN) is a wireless technology that has evolved to become a key for future networking. It is now an alternative technology that replaced standard wireless networking to play an important role for critical application and 99.9% uptime requirement. Using the Full-mesh topology in SECTION 2.1, Wireless Mesh Network can be fully utilized to its extent to achieve a high reliability networking system.

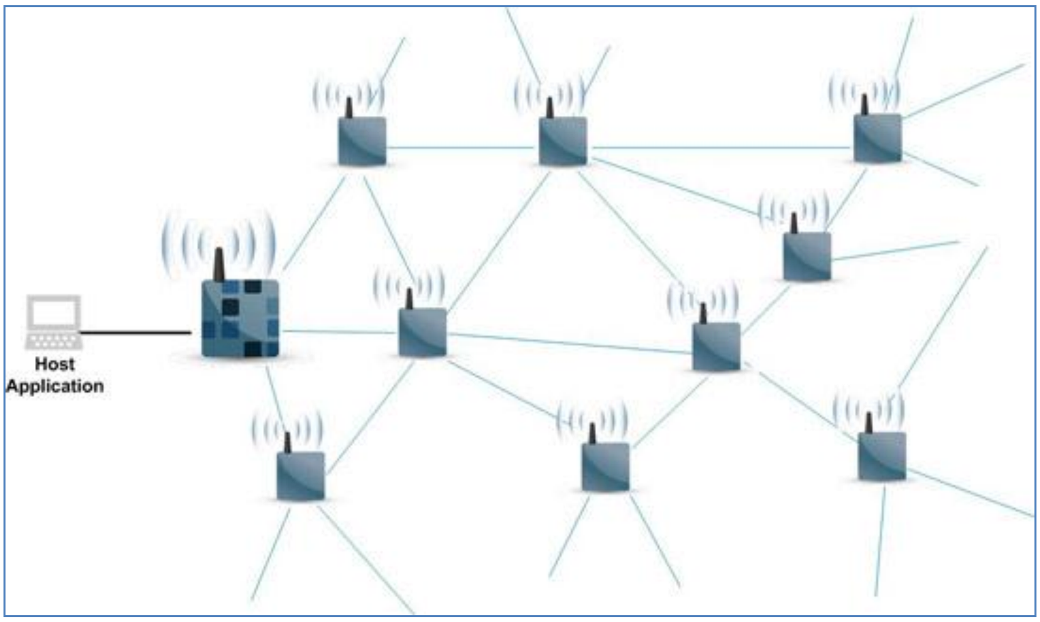


Figure 5 Wireless Mesh Topology (Source: [11])

The system is self-forming, they can automatically form a network whenever there are bunch of node in their neighbourhood that has the same group identification. They communicate between them without the need of human configuration.

The communication does not limited to point-to-point line of sight, it can be more than one hopping for a device to communicate with one another. This greatly increase coverage without the need to increase transmit power of wireless signal, it is called end-to-end communication, one device that resides in bottom-end can communicate with the one at the top-end through multi-hopping technique that will find the intermediate node to forward the data. For example in Figure 1, the right-sided device send all the information to the base station at the left, for data to arrive at the base station, it need to through multiple intermediate device that also act as a normal node.

The technology is designed in such a way that it can heal itself when undesired situation occurred. Let say that after the network is established properly, when there is one node fail to operate, the other node that depend on the faulty node to communicate can re-established a new network through other available path.

This multi-hop design will greatly reduce power consumption as the node does not need to transmit a long range wireless signal in order to send data to a far away node. As a result, a wide coverage can be achieved through lower transmission power [12].

2.3 Recommended Design and Requirement

In choosing the correct topologies and design for Robust Wireless Mesh Sensor Network that we will apparently use it for critical purpose, these are the design keys that we will be using:

1. Reliability – the network need to be nonetheless fault tolerant, the failure of any device/node will not affect the any other node [9]. The message/data need to be guaranteed to be delivered to its destination regardless of the situation. Network congestion is highly needed to be avoided [8].
2. Scale of the network – the scalability of network must be wide to compensate with the random placement of sensor in tough geographical area [8].
3. Flexibility – modification of network can be easily configured such as removal and addition of new sensors without the need of reconfiguration. The sensor also can be easily movable to a new destination [9].
4. All the sensors must be battery-powered as power source in a tough environment cannot be guaranteed. Although solar energy source may be implemented.
5. The latency – message need to be delivered in an acceptable time period to the monitoring base station.

CHAPTER 3

METHODOLOGY

In general, research methodology refers to a set of procedures used to conduct a research project. In here, the methodology includes:

- Research Methodology
- Project Activities
- Key Milestone
- Gantt Chart
- Tools

3.1 Research Methodology

Flowchart for the research methodology planned is shown in Figure 6.

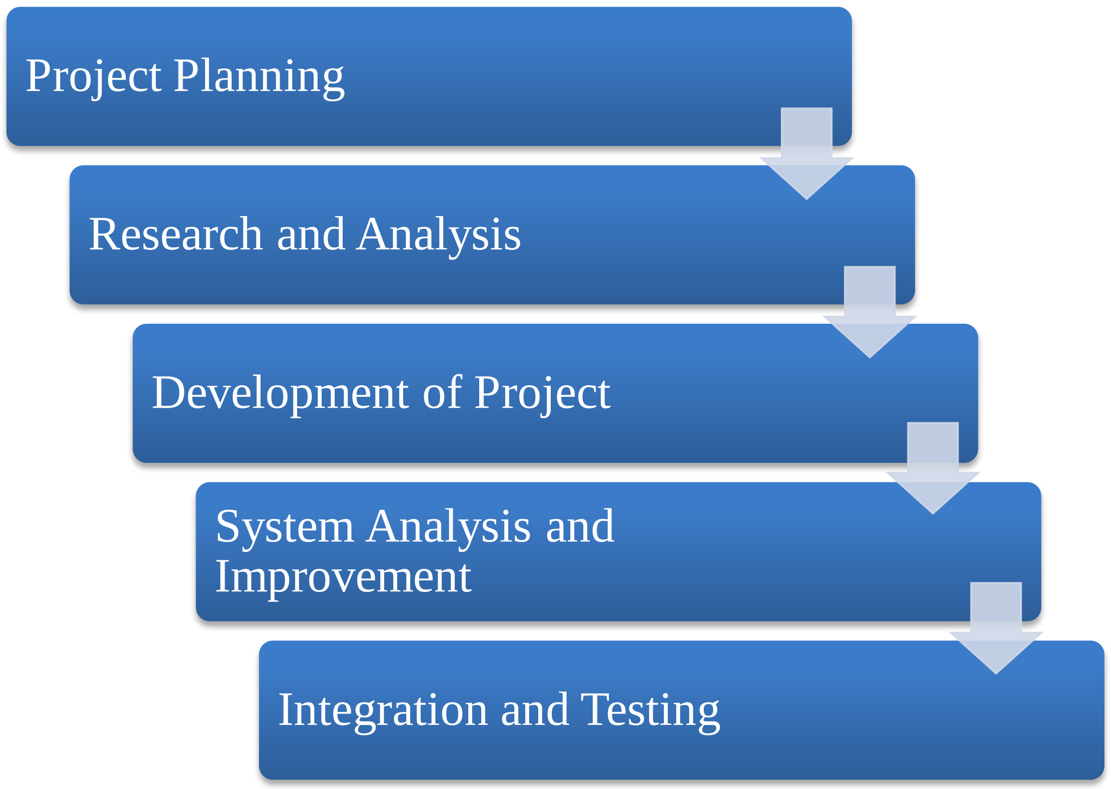


Figure 6 Research Methodology

3.1.1 Project Planning

Project planning is done by allocating task that are going to be done within the given time period. Careful planning is important to make sure this project can be carried out perfectly and meeting the requirement. A Gantt chart has been drawn which consist of several milestone and project activities so that the time will be allocated in the right way.

3.1.2 Research and Analysis

For this phase it involve the review of related journals, books, research papers and developers forum to increase the familiarity, better understanding and also to get a clear view about the research scope that will be carried out. The main information resources are from the ACM Digital Library and also IEEE.org. The existing survey and experiment done on the scope of the project is a vital source to analyze the system that will be developed later on. The existing system will be studied in order to find the weakness or certain features that they are lacking off so that it can be improved and tailored the new system based on the requirement.

3.1.3 Development of the Project

The project will be developed according to the default setting acquired from the hardware’s manufacturer. This will involve hardware interfacing and programming to achieve a standard goal for Wireless Sensor Network.

3.1.4 System Analysis and Improvement

The developed system is to be thoroughly studied and analysis is to be done to further understand how the system works in real environment. Thus further improvement can be done to really meet or exceed the requirement of the system. The system can possibly be narrowed down or specially designed for specific application.

3.1.5 Integration and Testing

After the improved system and code has been done and implemented, it will be tested again multiple times in the field with different quantities and situation. This is to see whether the system meet their requirement and expectation or not. The system will be carefully monitored for any inconsistencies.

3.2 Project Activities

In the beginning of the project, everything is focused on the theoretical reading and understanding the project scope. In this stage, critical analysis of the existing features of Wireless Sensor Network need to be done in order to find the features that has not being develop yet and also meet the requirement based on most applications.

The Wireless Sensor Network then will be developed using the hardware available to achieve a basic working WSN. To begin, there are always many tutorials on the internet as guidance in order to understand how the whole process works provided on the net and forum. The manuals given by the manufacturer will help a lot in the programming part. The devices are programmed accordingly until it meets the minimum requirement of a basic system.

Next phase will be the testing phase of the deployed system earlier. This includes the observation of the system for its latency, reliability, error-free and power consumption. Then, using the results, more research may be required to improve the characteristic of the deployed system. Continue developing the system, part by part and testing the features so that its meet the targeted requirement and also running as it supposed to be. The whole process need to be repeated while developing more other part until all the requirements are satisfied or reaching the time constraint given.

After completing the development and improvement phases, the system needs to undergo a thorough test in the field to see its true reliability and robustness for final report of the working system. If required, the system may be redevelop or reconfigure when there are still time available.

3.3 Key Milestone

Below are the key milestone that need to be achieved for the project development throughout both of the semester of Final Year Project 1 (FYP I) and Final Year Project 2 (FYP II).

Semester 1

Table 1 Key Milestone for FYP I

Milestone	Week
Project Planning	Week 6
General Research	Week 9
Programming Study	Week 11
Development of basic WSN	Week 13

Semester 2

Table 2 Key Milestone for FYP II

Milestone	Week
Analysis on the deployed basic WSN	Week 5
Improvement on the deployed system	Week 9
Testing on the improved system	Week 12
Final system integration	Week 14

3.4 Gantt Chart

Table 3 Gantt chart

Phase		**FYP 1**															**FYP 2**														
		Weeks																													
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	Project Planning	▨	▨	▨	▨	▨	▨																								
2	General Research				▨	▨	▨	▨	▨	▨																					
3	Study of Programming Language									▨	▨	▨																			
4	Development of basic WSN											▨	▨	▨																	
5	Analysis on the deployed basic WSN																	▨	▨	▨	▨										
6	Improvement on the deployed system																			▨	▨	▨	▨	▨	▨						
7	Testing on the improved system																									▨	▨	▨			
8	Final system integration																												▨	▨	

3.5 Tools

The tools that need to be used in this project will involve several hardware and software. These are the hardware:

1. XM2110CA – 2.4GHz IRIS OEM Reference Board

   This is the main board from MEMSIC Inc. that functions as the CPU and transceiver in a single board.

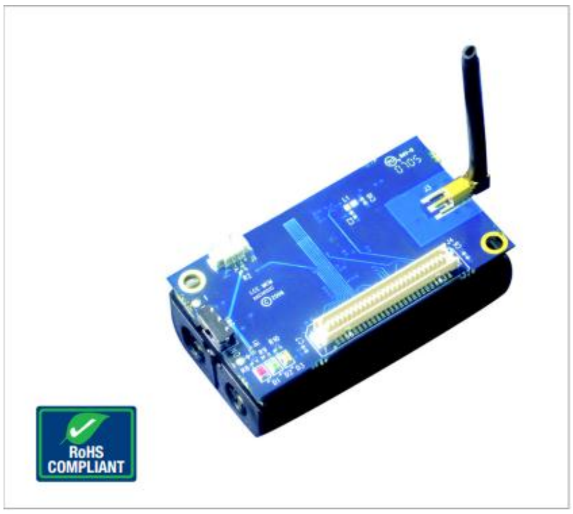


Figure 7 IRIS Mote (Source : MEMSIC datasheet)

2. MTS 420CC – Weather Sensor Board with Light, Temperature, Humidity, Barometric Pressure, Seismic and GPS.

   This device manufactured by MEMSIC Inc. can be interfaced with the IRIS Mote above to function as a sensor node. It wide varieties of sensors in one board make it applicable for many applications.

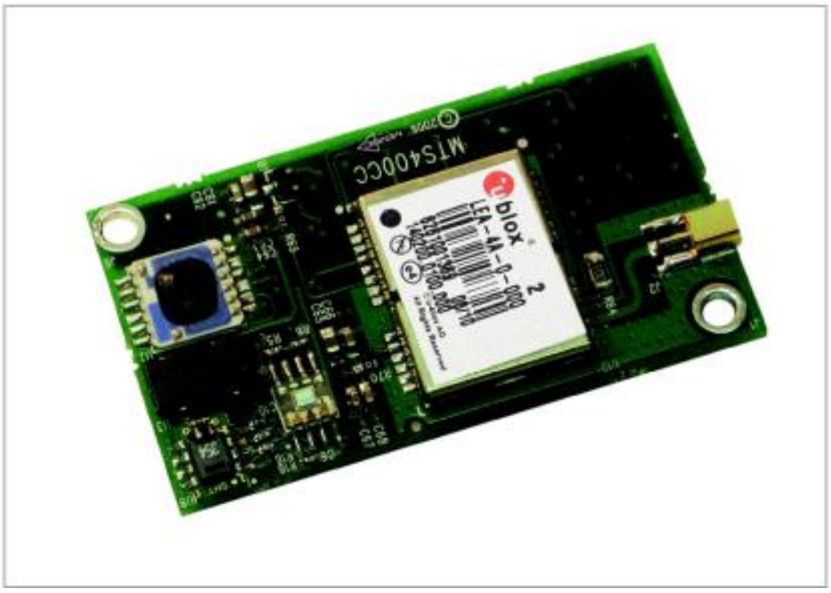


Figure 8 MTS420CC (Source: MEMSIC datasheet)

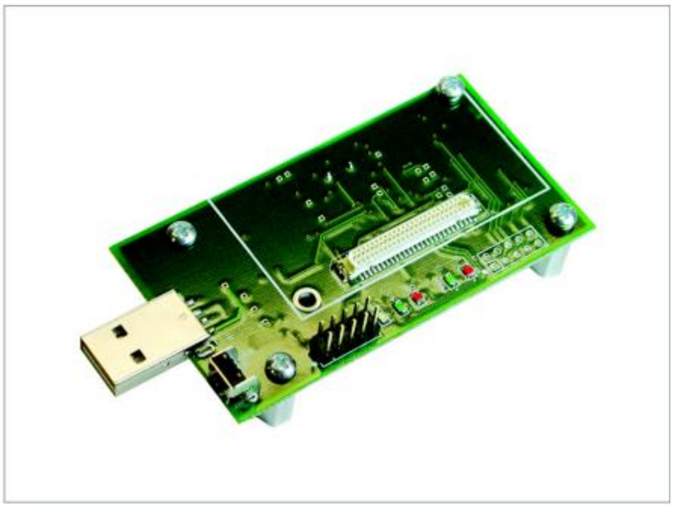


Figure 9 MIB520CB (Source: MEMSIC datasheet)

3. MIB520CB – USB PC Interface Board

   The board is also from MEMSIC Inc., designed as plug-and-play with other MEMSIC's board to add the gateway function in the node. This board will allow the node to interface with PC via USB connection to be programmed or acts as a base station.

All the software needed can be obtained from the MEMSIC itself, including the programming software, monitoring program and also manuals. The software products that will be used are:

1. Moteworks

   It is a software which enables user to compile code and build it into the hardware. It includes all the libraries that may be used to compile the code. It uses NesC compiler which is the extension of C language. This language will then being implemented in the operating system of the hardware used, TinyOS [**13**].

2. Cygwin

   Linux virtual platform for Windows, it is needed for TinyOS compiling and pprogramming tools to run normally.

3. Xsniffer

   Monitor the basic mote's network parameter. Xsniffer is compatible with MEMSIC's OEM hardware.

4. Moteview

   Moteview is an interfacing software that connects with the database and motes for easy and interactive mote monitoring at the "end-user" side. This package is included with other tools such as Xserve (command line tools that Moteview GUI run in background to capture packet through the serial interface).

3.6 Project's Preparation

This project was started with the hardware's assembling and configuration. Through the hardware manual, the devices can be assembled easily by mean of plug-and-play. To be accurate, there were three types of devices, the processor and RF module (mote), the sensor boards and lastly the interface board as described in the Tools (SECTION 3.5) above.

3.6.1 Installing the required software (Moteworks and XMESH)

The first basic things to do was installing the given software by MEMSIC, two major packages were given:

- Moteworks
- Moteview

Initially, it is compulsory for users who are going to use the software given by MEMSIC to have Windows XP SP3 or older operating system for the software to works. Thus installing one is a need. After installing both software, there are problems that usually rise after that, you can solve it by installing and reinstalling various version of .NET Framework from Microsoft's website. Usually the programs will run properly with .NET Framework 1.1 given in the manufacturer's CD.

These are the additional component for the software to work and all of these were given in the installation CD.

• PostgreSQL 8.0 database service

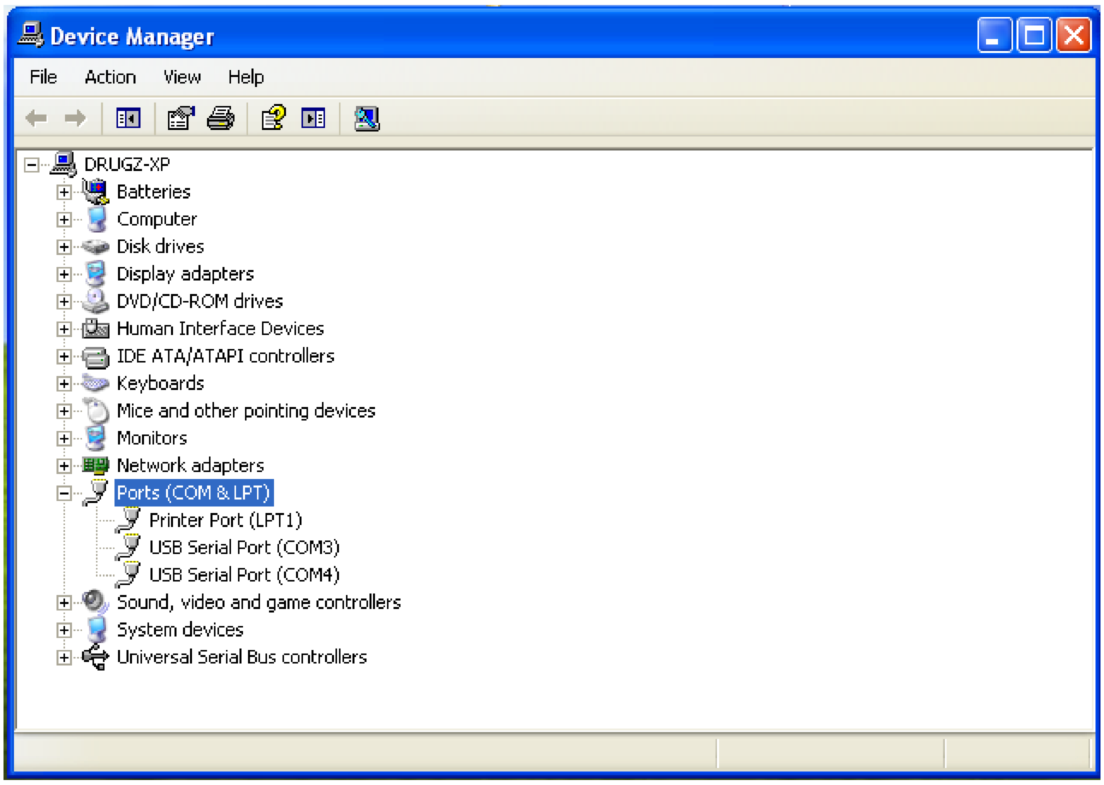
• PostgreSQL ODBC driver

• Microsoft .NET 1.1 framework

As we are purchasing the USB type of interface gateway, MEMSIC's MIB520CB, it needs an additional driver for the board to working properly. This driver will convert the USB port in a PC to serial interface port, to be precise, two serial ports. To check whether the driver was working properly or not, go to

Start>Control Panel> Device Manager

Check under Ports (COM & LPT), make sure there were two additional USB Serial Port as in Figure 10 below. In this case, they were COM3 and COM4.



DRUGZ-XP
- Batteries
- Computer
- Disk drives
- Display adapters
- DVD/CD-ROM drives
- Human Interface Devices
- IDE ATA/ATAPI controllers
- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
- Ports (COM & LPT)
  - Printer Port (LPT1)
  - USB Serial Port (COM3)
  - USB Serial Port (COM4)
- Sound, video and game controllers
- System devices
- Universal Serial Bus controllers

Figure 10 Two additional USB Serial Ports

Please be noted that the some of the programs cannot be run unless the interface board is already connected. Actually, I have tried running the software before connecting the interface board, it gives an unknown error then the software terminated. I thought that the software was not installed correctly but after the problem was solved when the software was run while the device connected.

3.6.2 Hardware Parts Assembly

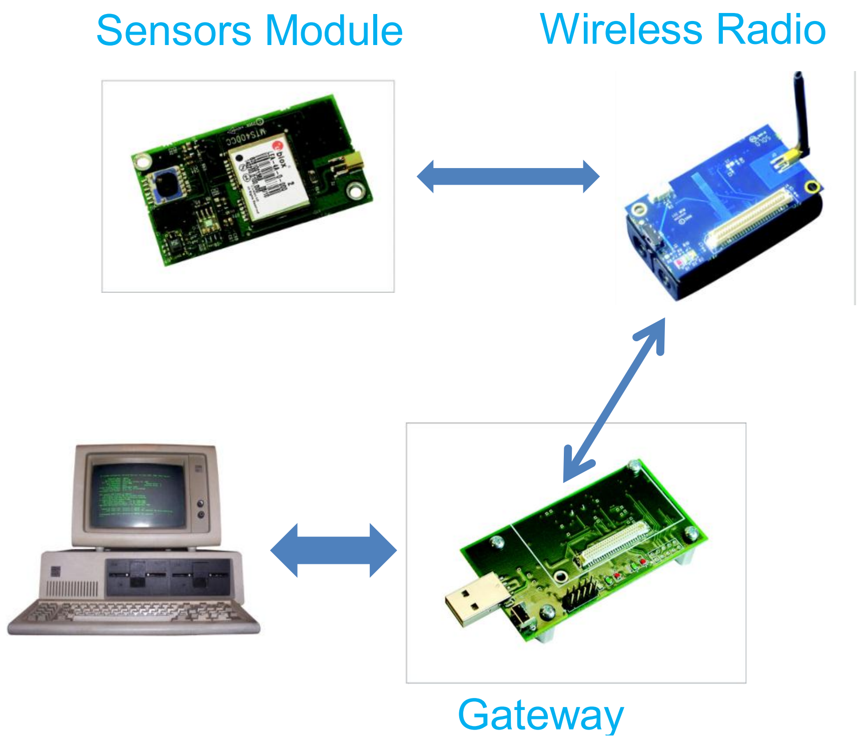


Figure 11 Hardware Configuration

Gateway base can be assembled by connecting the main RF/CPU board (IRIS) to the interface board, in this case, the IRIS with the USB Interface board. Figure 11,12 and 13 show the assembled boards.

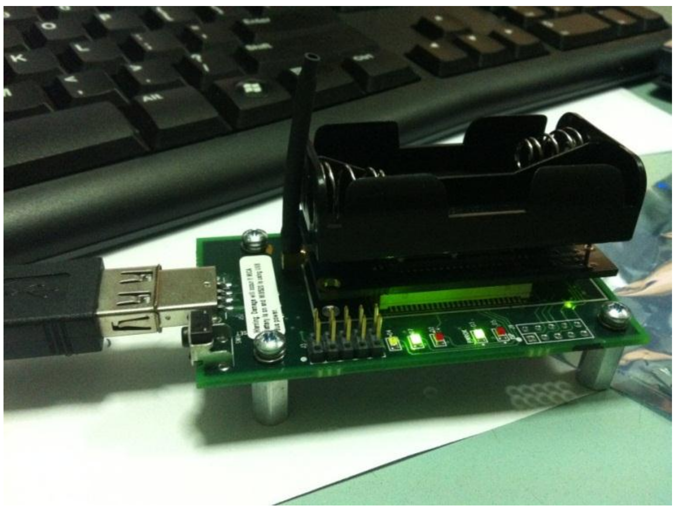


Figure 12 Assembled gateway

The sensor node can be deployed by connecting an IRIS with a sensor board. In Figure13, the sensor board used was MTS420CC.

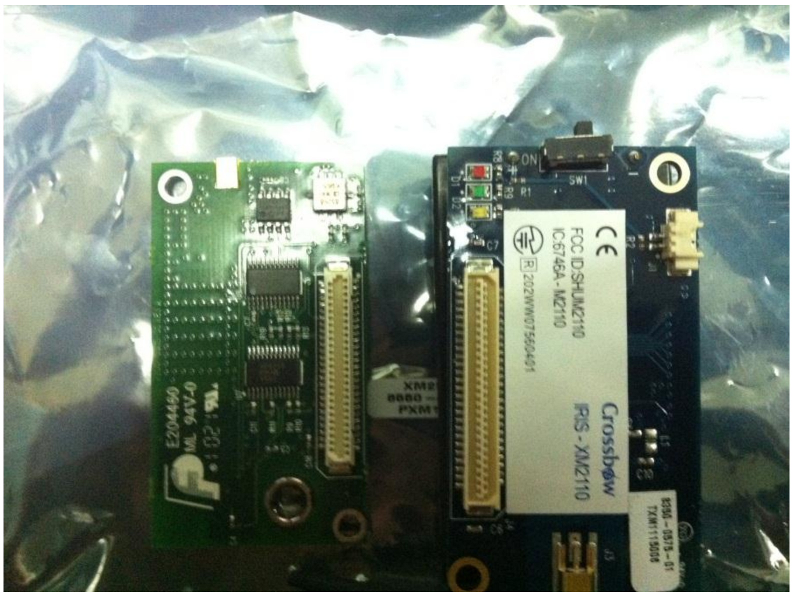


Figure 13 IRIS and MIB520

3.6.3 Programming the Device using Moteworks

Four icons will showed up at the desktop after the software packages were installed. They were Xniffer 1.0, PN2, MoteConfig 2.0 and Moteview 2.0F. These were the main program that will be used for the software.

In order to program a certain code into the processor on the board, there are actually lot of things to be done before the codes are transferred to the processor, all of them are commands in linux (Cygwin). Thanks to the software provided by MEMSIC, it is now simplified by just writing the code, compile and build to binary and fusing them onto the processor's memory.

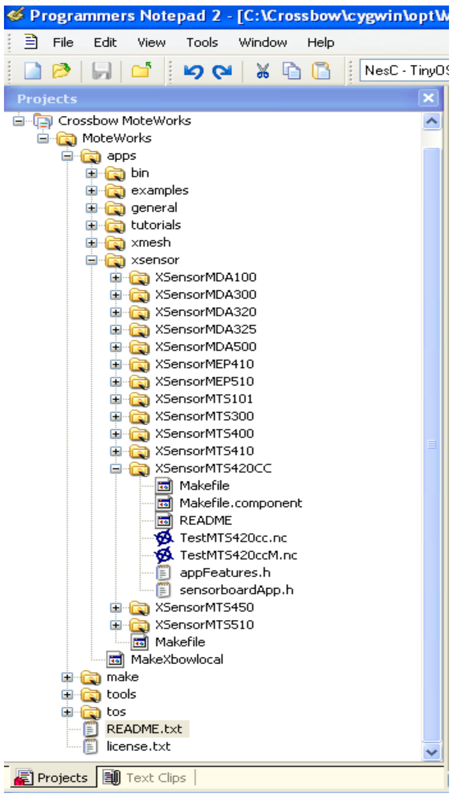
The first step is to open the PN2 (Programmers Notepad 2), it is a software in windows that are indirectly connected with Cygwin platform to do the compilation of codes. The language of the TinyOS operating system is mostly the same as C-embedded language, information can be search in the manual given and from the Internet.

MEMSIC has provided its own library and test code for TinyOS in deploying the Mesh Sensor Network, which they called it XMesh. Inside the Cygwin Project Files, the code are prebuilt there as in Figure 14, Using the provided library, these code is written to make the devices function as a gateway/base station that will monitor the other nodes and acquire data from them and also forwarding packets. This code uses the default device parameter declared in a file called “MakeXbowlocal”. Some of the important parameter in it are:

- Radio Channel

- RF Power Level

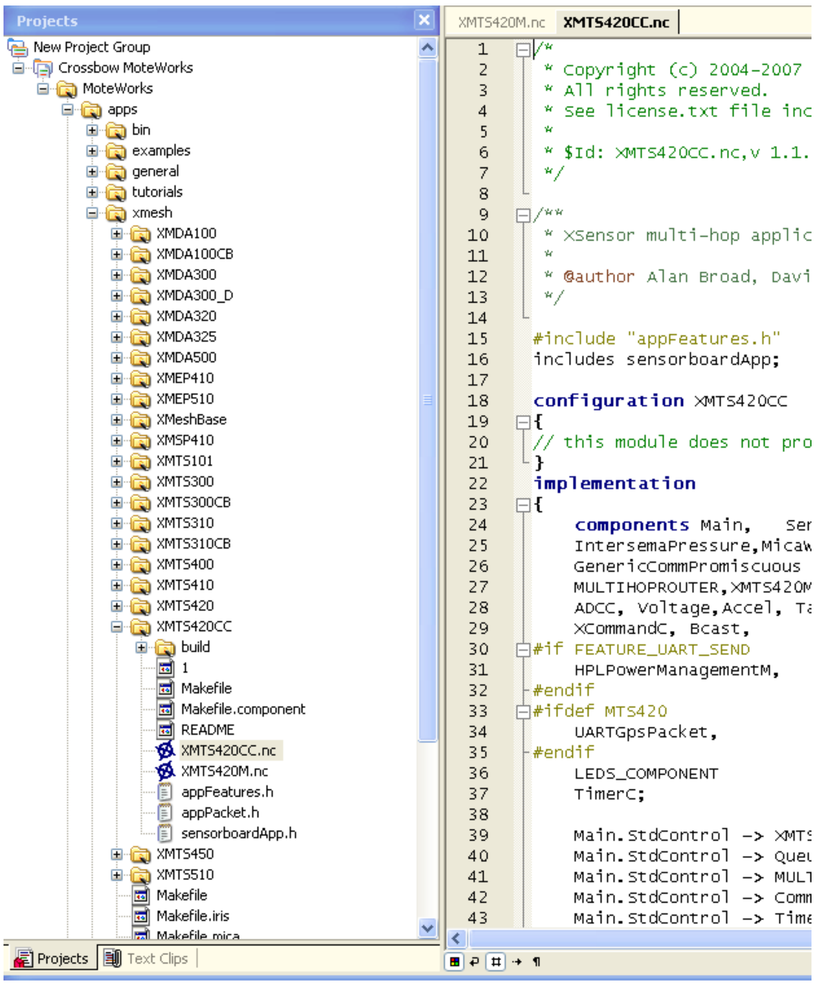
- Group ID



```
Crossbow MoteWorks
  MoteWorks
    apps
      bin
      examples
      general
      tutorials
      xmesh
      xsensor
        XSensorMDA100
        XSensorMDA300
        XSensorMDA320
        XSensorMDA325
        XSensorMDA500
        XSensorMEP410
        XSensorMEP510
        XSensorMTS101
        XSensorMTS300
        XSensorMTS400
        XSensorMTS410
        XSensorMTS420CC
          Makefile
          Makefile.component
          README
          TestMTS420cc.nc
          TestMTS420ccM.nc
          appFeatures.h
          sensorboardApp.h
        XSensorMTS450
        XSensorMTS510
        Makefile
      MakeXbowlocal
    make
    tools
    tos
    README.txt
    license.txt
```

Figure 14 Prebuilt Library of Motework

Without changing those basic device parameters, this code will perform at its basic level of Mesh networking with default parameters. The sensorboard application can be chosen from the left pane in the Programmer’s Notepad as in Figure 15.



```
/*
 * Copyright (c) 2004-2007
 * All rights reserved.
 * See license.txt file inc
 *
 * $Id: XMTS420CC.nc,v 1.1.
 */

/**
 * XSensor multi-hop applic
 *
 * @author Alan Broad, Davi
 */

#include "appFeatures.h"
includes sensorboardApp;

configuration XMTS420CC
{
// this module does not pro
}
implementation
{
    components Main,    Ser
    IntersemaPressure,MicaW
    GenericCommPromiscuous
    MULTIHOPROUTER,XMTS420M
    ADCC, Voltage,Accel, Ta
    XCommandC, Bcast,
#if FEATURE_UART_SEND
    HPLPowerManagementM,
#endif
#ifdef MTS420
    UARTGpsPacket,
#endif
    LEDS_COMPONENT
    TimerC;

    Main.StdControl -> XMTS
    Main.StdControl -> Queu
    Main.StdControl -> MULT
    Main.StdControl -> Comm
    Main.StdControl -> Time
```

Figure 15 Choosing the sensorboard in Xmesh

To build and install the code into device, a command parameter needs to be executed. The command line (linux shell) can be access using the Programmer note program or through Cygwin itself. The command can be executed directly through the software by going to Tools>Shell or by pressing F6. The basic command syntax is as below:

To install into IRIS using the USB Interface MIB520

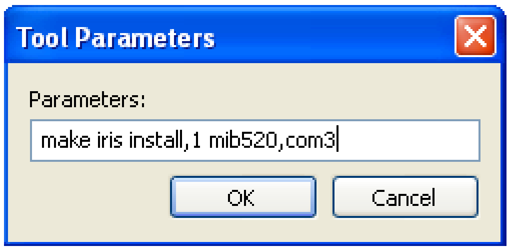
make iris install,*<node id>* MIB520,com*<x>*

Where <x> is the lower USB serial port number that we saw from the Device Manager earlier.

Example (Figure 16),

make iris install,1 MIB520,com3

This will build the code and then transfer it into the device through MIB520 COM port 3.



Tool Parameters

Parameters:

```
make iris install,1 mib520,com3
```

Figure 16 Shell tools in Programmer's Notepad

3.6.4 Programming the TinyOS in Linux Environment

The necessity to use TinyOS in Linux comes in handy when the programmer wants to compile the nesC application with Java for communication purpose. Furthemore, in the analysis later in this project, the author decided to use TinyOS 2.1.0 instead of Moteworks programming environment as the manufacturer did not provide enough source code to alter the routing protocol on network layer.

The TinyOS tools provide the easiest way to communicate the mote with PC through Serial or USB port., and most of these communication tools uses Java in its compilation. Cygwin can also be used when compiling nesC and Java but a lot of problems may arise depending on your luck.

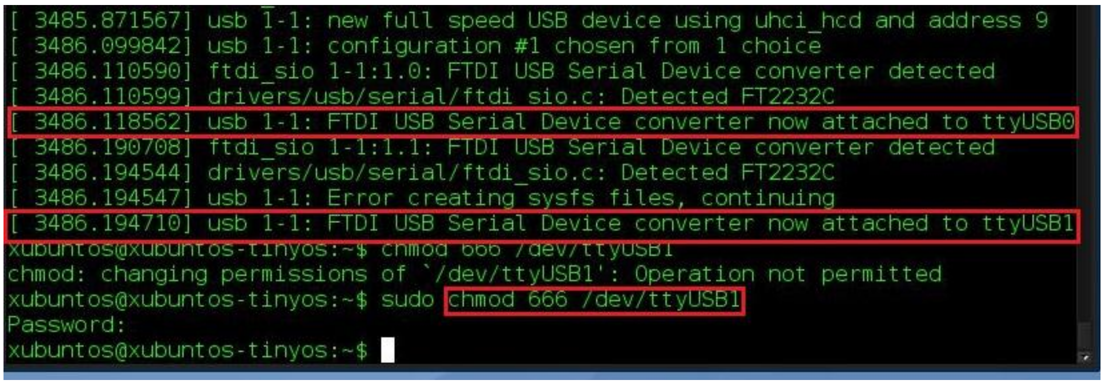
The easiest way to use a machine with Linux and Java installed is to use a preconfigured VMWare image provided by the TinyOS Community. This image file and the setup instruction can be referred in [**14**].

After installing and running the pre-configured Linux XUBUNTOS by going through the guideline in [**14**], connect a mote to test its connection. The author used MIB520 programming board which has USB-to-Serial converter. Invoking **motelist** will return no result. In order to solve this problem, another method can be used, open a terminal and type :

```
~$ dmesg
```

The result of **dmesg** is shown in Figure 17.

The mote connected can be seen as ttyUSB*n* (e.g. ttyUSB0 and ttyUSB1), *n* can be different depending on port usage. Thus to address the port used in Linux, */dev/ttyUSBn* need to be used instead of *COMn*. Noted that in the Figure 17, there are two USB port used.



```
[ 3485.871567] usb 1-1: new full speed USB device using uhci_hcd and address 9
[ 3486.099842] usb 1-1: configuration #1 chosen from 1 choice
[ 3486.110590] ftdi_sio 1-1:1.0: FTDI USB Serial Device converter detected
[ 3486.110599] drivers/usb/serial/ftdi_sio.c: Detected FT2232C
[ 3486.118562] usb 1-1: FTDI USB Serial Device converter now attached to ttyUSB0
[ 3486.190708] ftdi_sio 1-1:1.1: FTDI USB Serial Device converter detected
[ 3486.194544] drivers/usb/serial/ftdi_sio.c: Detected FT2232C
[ 3486.194547] usb 1-1: Error creating sysfs files, continuing
[ 3486.194710] usb 1-1: FTDI USB Serial Device converter now attached to ttyUSB1
xubuntos@xubuntos-tinyos:~$ chmod 666 /dev/ttyUSB1
chmod: changing permissions of `/dev/ttyUSB1': Operation not permitted
xubuntos@xubuntos-tinyos:~$ sudo chmod 666 /dev/ttyUSB1
Password:
xubuntos@xubuntos-tinyos:~$
```

Figure 17 "dmesg" terminal result

CHAPTER 4
RESULTS AND DISCUSSION

4.1 Data Gathering and Analysis

The first analysis conducted is to find the approximate range of the mote's radio. The overview of the experiment is as in Figure 18.

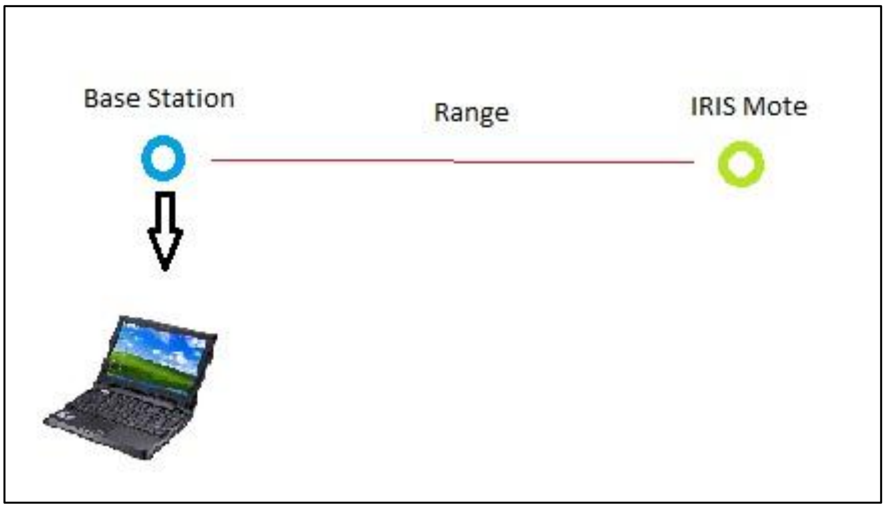


Figure 18 Open Field Test

The devices are placed in a field with no obstruction to interfere with the device's radio. The results are shown in Table 4.

Table 4 IRIS's Radio Transmission Open Field Range

Transmission Power (dBm)	Range (m)
-17	11.4
-12	32.2
-9	47.6
-7	67
-5	70
-4	70
-3	74
-2	79
-1	80
0	81
0.7	85
1.3	96
1.8	120
2.3	163
2.8	193
3.2	241

The overview of the results then is shown by plotting the graph of Range vs Transmission Power as in Figure 19.

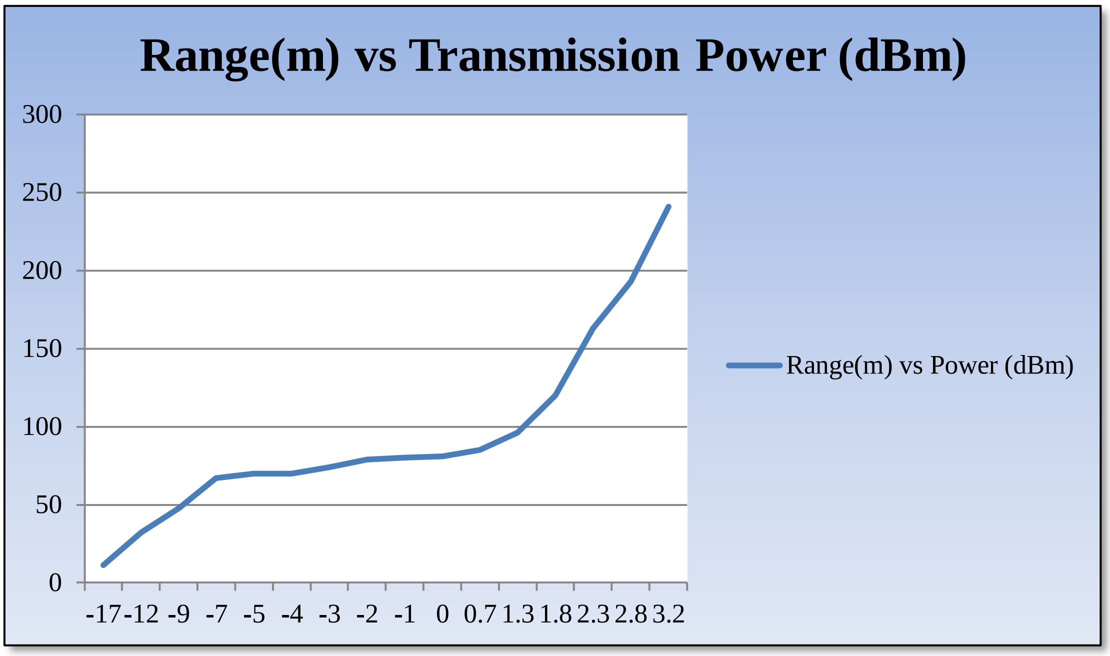


Figure 19 Range vs Transmission Power

These ranges are crucial for further experiment of the device as to test a working multi-hop network with mesh topology, the nodes are required to be in a certain range for specified transmission power.

4.2 Modelling and Experimentation

MEMSIC has already provided their basic working Multi hop Mesh Network that includes its own routing protocol called Xmesh. This routing algorithm is mainly used because it is more reliable compared to other routing protocols.

Generally explain, Xmesh uses a different method in calculating the route paths to the base station instead of traditional distance vector routing cost metric used by other algorithm. Distance vector routing is basically a hop count, which means that a node will send its data to the base station through the shortest path by means of the least hop required. While Xmesh uses its owh cost metric, called Minimum Transmission (MT), it takes account of link quality in the calculation to avoid excessive use of energy as transmission of data through low quality link will require more power [**15**].

Xmesh can be built and flashed into the mote through the Programmer's Notepad of Moteworks or MoteConfig.

4.2.1 Wireless Sensor Network with XMESH Algorithm

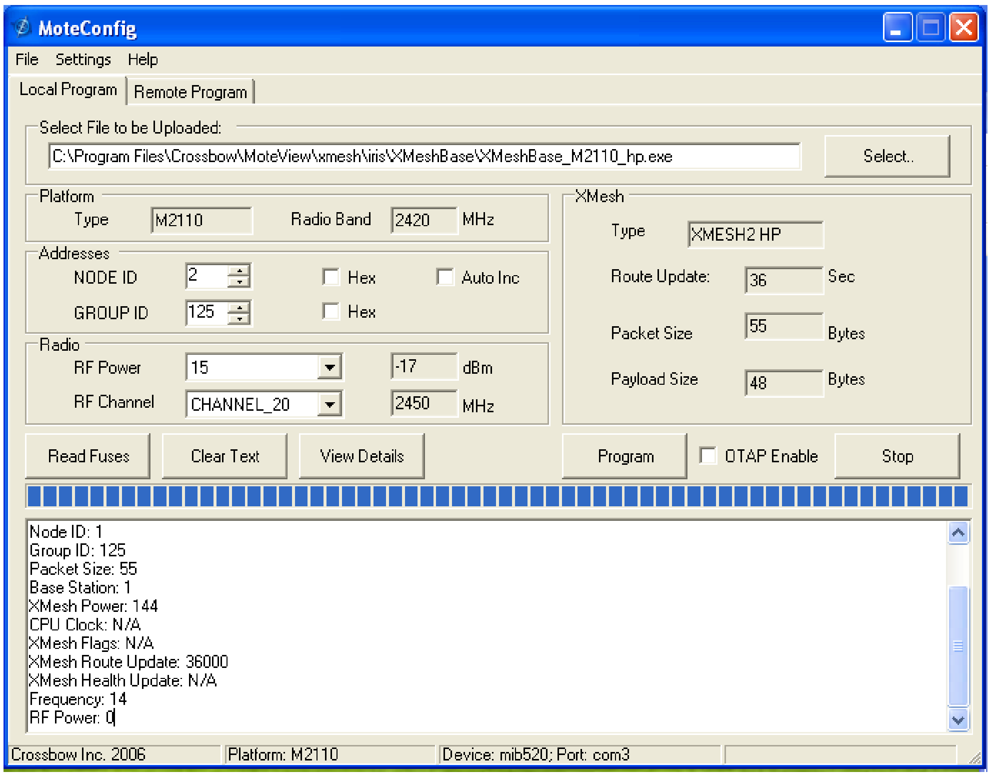
Deployment of XMESH routing algorithm can be done with ease. This next code will control the sensor components inside the sensor board and communicate them with the CPU to be transmitted to nearby node with the same group ID.

Base station can be programmed using the provided code in:

Moteworks>apps>xmesh>XmeshBase

Another approach in compiling XMesh into the mote can be done by using the provided MoteConfig program and the Xmesh binary file can be found in: (Figure 20)

C:\Program Files\Crossbow\Moteview\Xmesh



Select File to be Uploaded: C:\Program Files\Crossbow\MoteView\xmesh\iris\XMeshBase\XMeshBase_M2110_hp.exe

Platform
Type M2110 Radio Band 2420 MHz

Addresses
NODE ID 2 Hex Auto Inc
GROUP ID 125 Hex

Radio
RF Power 15 -17 dBm
RF Channel CHANNEL_20 2450 MHz

XMesh
Type XMESH2 HP
Route Update: 36 Sec
Packet Size 55 Bytes
Payload Size 48 Bytes

```
Node ID: 1
Group ID: 125
Packet Size: 55
Base Station: 1
XMesh Power: 144
CPU Clock: N/A
XMesh Flags: N/A
XMesh Route Update: 36000
XMesh Health Update: N/A
Frequency: 14
RF Power: 0
```

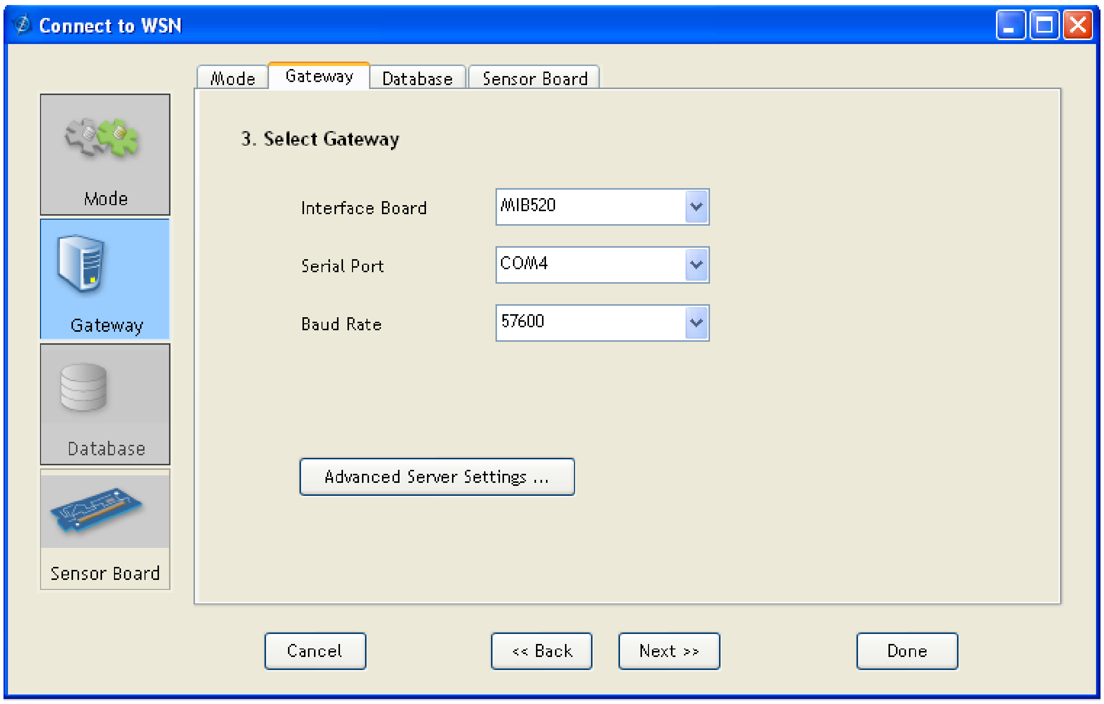
Crossbow Inc. 2006 | Platform: M2110 | Device: mib520; Port: com3

Figure 20 MoteConfig

4.2.2 Visualizing the programmed nodes through MoteView

Moteview was used to test the programmed node as it is a software that enable us to monitor the data that the base station received. Moteview functions by communicating with the database and the linux environment platform. The data from the base station comes into the database through Xserve, a program in Cygwin linux serve as the interface between device and database. This Xserve will then insert strings of data into the database. The database used is PostgreSQL and it is installed in Microsoft Windows whereas the Xserve is built under Cygwin linux.

Double clicking the Moteview icon in the desktop will open the program. The interface board need to be connected all the time during this time. Click “Connect to WSN” icon in the program, then these setting as in Figure 21 is introduced to connect to the base station.

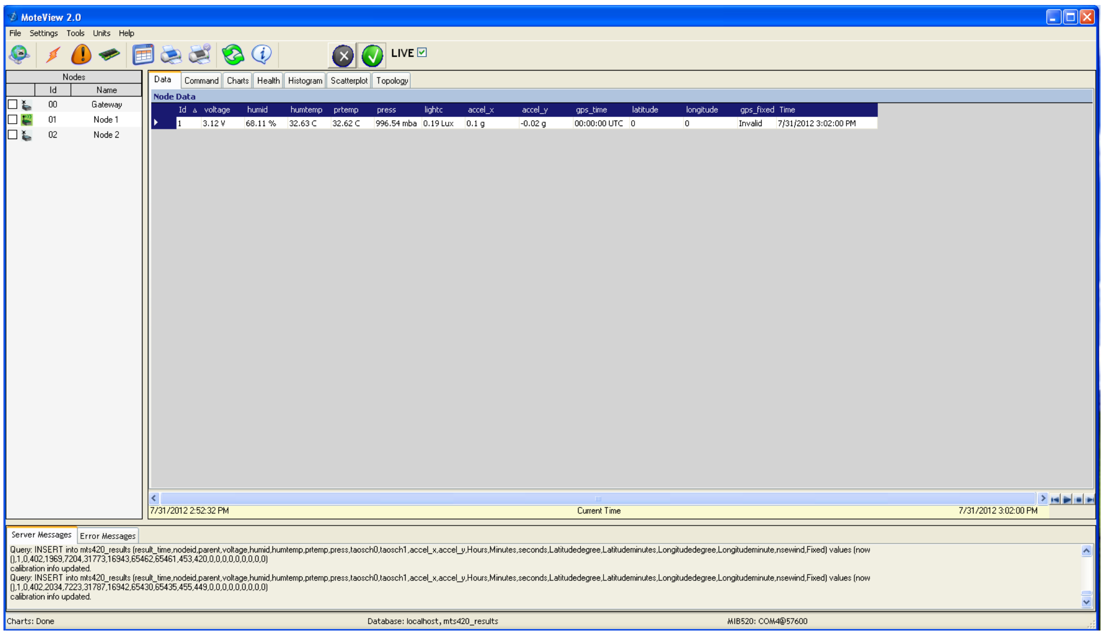


3. Select Gateway

Interface Board	MIB520
Serial Port	COM4
Baud Rate	57600

Figure 21 Connecting to WSN using Moteview

These parameters are for MIB520 with the second virtual USB Serial Port is COM4. By clicking the “Tick” symbol on top right of the software, the monitoring will begin. Figure 22 below shows the programs functioning in recording the data from base station.



Nodes

Id	Name
00	Gateway
01	Node 1
02	Node 2

Node Data

Id	voltage	humid	humtemp	prtemp	press	lightc	accel_x	accel_y	gps_time	latitude	longitude	gps_fixed	Time
1	3.12 V	68.11 %	32.63 C	32.62 C	996.54 mba	0.19 Lux	0.1 g	-0.02 g	00:00:00 UTC	0	0	Invalid	7/31/2012 3:02:00 PM

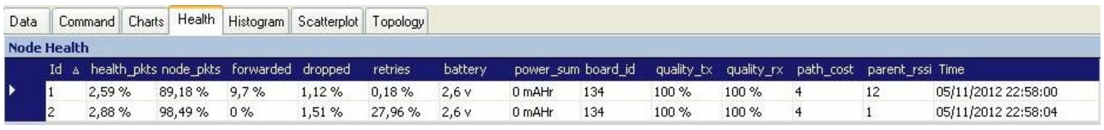
7/31/2012 2:52:32 PM Current Time 7/31/2012 3:02:00 PM

```
Query: INSERT into mts420_results (result_time,nodeid,parent,voltage,humid,humtemp,prtemp,press,taosch0,taosch1,accel_x,accel_y,Hours,Minutes,seconds,Latitudedegree,Latitudeminutes,Longitudedegree,Longitudeminute,nsewind,Fixed) values (now
(),1,0,402,1969,7204,31773,16943,65462,65461,453,420,0,0,0,0,0,0,0,0,0)
calibration info updated.
Query: INSERT into mts420_results (result_time,nodeid,parent,voltage,humid,humtemp,prtemp,press,taosch0,taosch1,accel_x,accel_y,Hours,Minutes,seconds,Latitudedegree,Latitudeminutes,Longitudedegree,Longitudeminute,nsewind,Fixed) values (now
(),1,0,402,2034,7223,31787,16942,65430,65435,455,449,0,0,0,0,0,0,0,0,0)
calibration info updated.
```

Charts: Done Database: localhost, mts420_results MIB520: COM4@57600

Figure 22 Moteview test

The output message at the bottom shows the query derived from the database and it is updated in the table field. The experiment is repeated for two (2) motes that are placed further apart to introduce Mesh/Hop configuration. The results are as in Figure 23.


Data
Command
Charts
Health
Histogram
Scatterplot
Topology

Node Health

Id	health_pkts	node_pkts	forwarded	dropped	retries	battery	power_sum	board_id	quality_tx	quality_rx	path_cost	parent_rssi	Time
1	2,59 %	89,18 %	9,7 %	1,12 %	0,18 %	2,6 v	0 mAHr	134	100 %	100 %	4	12	05/11/2012 22:58:00
2	2,88 %	98,49 %	0 %	1,51 %	27,96 %	2,6 v	0 mAHr	134	100 %	100 %	4	1	05/11/2012 22:58:04

Figure 23 Moteview Health Data

As in Figure 21, some packets are forwarded (9.7%) by Node 1. This means that Node 2 is hopping through the Node 1 to reach the base station. The value 9.7% means that 9.7% of packets sent by Node 1 to the base station is the packets from Node 2 that are being forwarded.

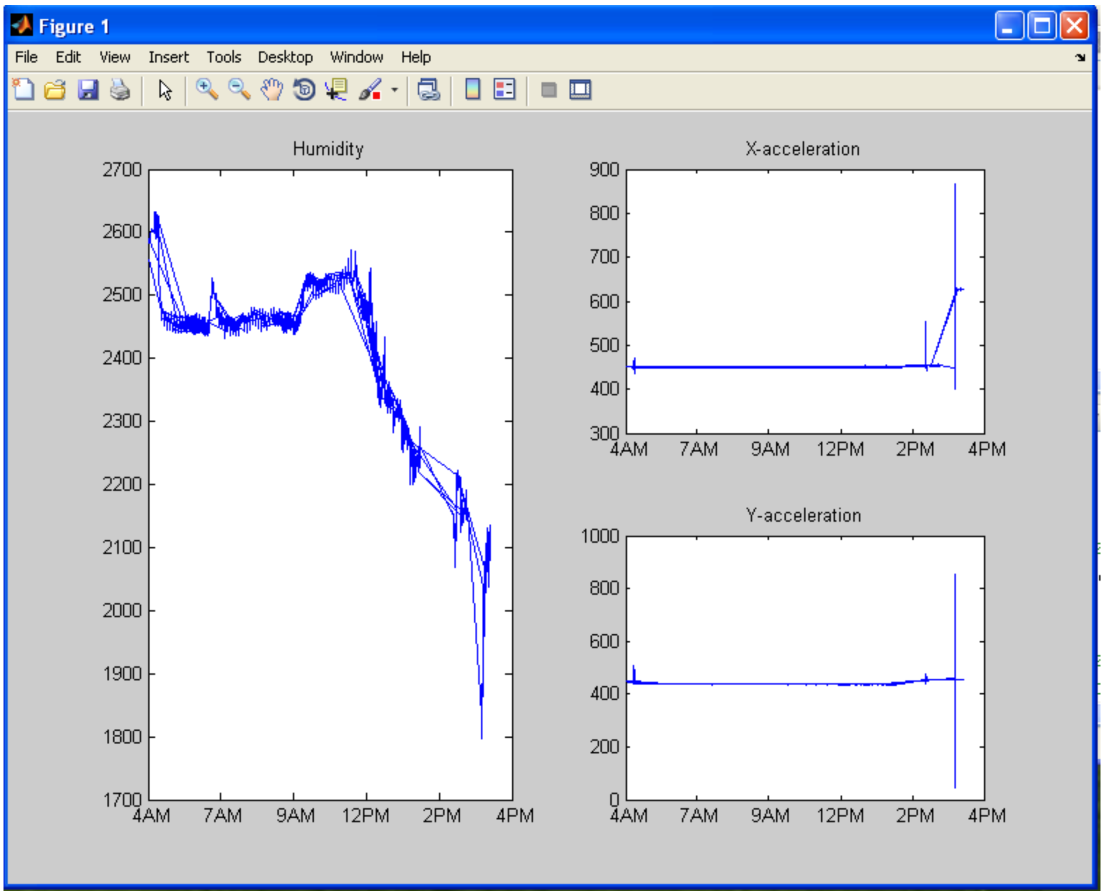


Figure 24 MATLAB Graph

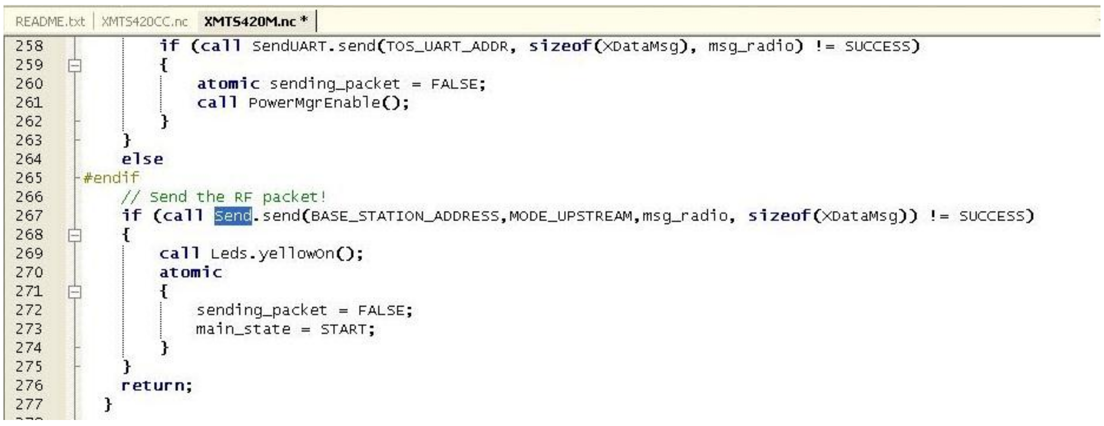
Instead of using the Moteview, the author has written a code in MATLAB that can read the data in the database and visualize it in the MATLAB way as in Figure 24. This code generally extract the data from the PostgreSQL database and plot it right away. As the matter does not concern much in this project, the development of the MATLAB code stop there.

The main advantage of using MATLAB instead of the given Moteview is the developer can extract anything he wants from the database. As in Moteview, only one type of sensor modules can be used for all the motes to properly parse the data sinked into the BaseStation. The MATLAB code can be referred in Appendix A

4.2.3 Analysis of XMESH routing in Moteworks

After going through the provided Xmesh code provided, there are no evidence found that can provide a way to look into the Xmesh Routing Algorithm. This means that the manufacturer did not provide all the source code of the Xmesh. The author then proceeds to change the programming method from using the provided Moteworks to the legacy TinyOS 2.x.

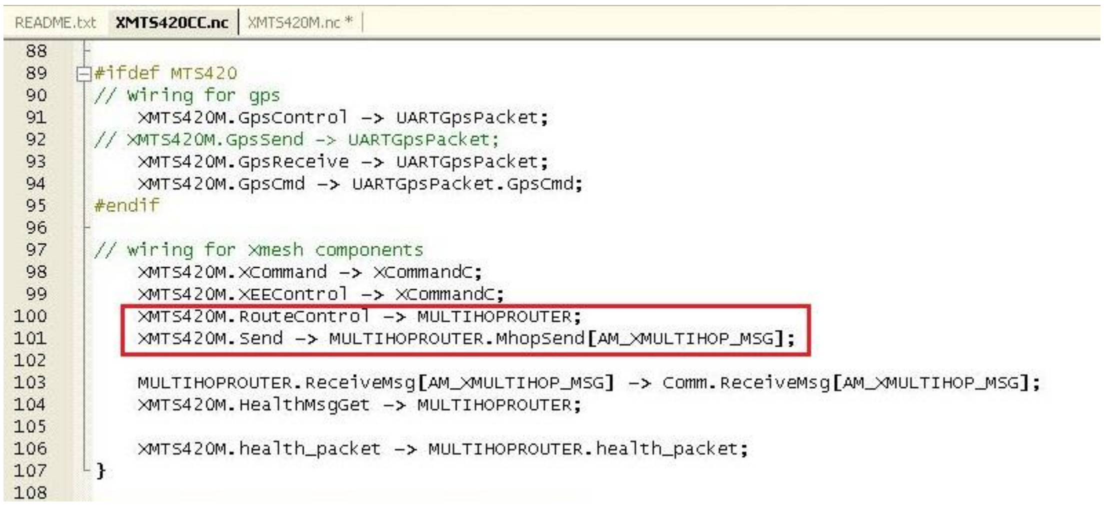
This problem can be seen by checking the wiring interfaces/components in the nesC code and its usage.



```
            if (call SendUART.send(TOS_UART_ADDR, sizeof(XDataMsg), msg_radio) != SUCCESS)
            {
                atomic sending_packet = FALSE;
                call PowerMgrEnable();
            }
        }
        else
#endif
        // Send the RF packet!
        if (call Send.send(BASE_STATION_ADDRESS,MODE_UPSTREAM,msg_radio, sizeof(XDataMsg)) != SUCCESS)
        {
            call Leds.yellowOn();
            atomic
            {
                sending_packet = FALSE;
                main_state = START;
            }
        }
        return;
    }
```

Figure 25 Xmesh Problem 1

Figure 25 shows that *Send* interface is used to send data to lower layer (network layer) for further network routing process. The lower layer source code can be found in *$TOSROOT/tos*.



```

#ifdef MTS420
// wiring for gps
    XMTS420M.GpsControl -> UARTGpsPacket;
// XMTS420M.GpsSend -> UARTGpsPacket;
    XMTS420M.GpsReceive -> UARTGpsPacket;
    XMTS420M.GpsCmd -> UARTGpsPacket.GpsCmd;
#endif

// wiring for xmesh components
    XMTS420M.XCommand -> XCommandC;
    XMTS420M.XEEControl -> XCommandC;
    XMTS420M.RouteControl -> MULTIHOPROUTER;
    XMTS420M.Send -> MULTIHOPROUTER.MhopSend[AM_XMULTIHOP_MSG];

    MULTIHOPROUTER.ReceiveMsg[AM_XMULTIHOP_MSG] -> Comm.ReceiveMsg[AM_XMULTIHOP_MSG];
    XMTS420M.HealthMsgGet -> MULTIHOPROUTER;

    XMTS420M.health_packet -> MULTIHOPROUTER.health_packet;
}

```

Figure 26 Xmesh Problem 2

Figure 26 further shows that the *Send* interface used by the main code is wired to *MULTIHOPROUTER* component, while this component can nowhere be founded in the TOS libraries.

Further research by the author has confirmed this matter, an email from Giri Baleri, a Crossbow individual itself stated that the source code of the Xmesh Routing is only available for Enterprise users. The email can be find here [**16**].

In the source code's directory tree of the author, there is XMeshBinary directory exists, this confirm that the version that the author got is the standard version.

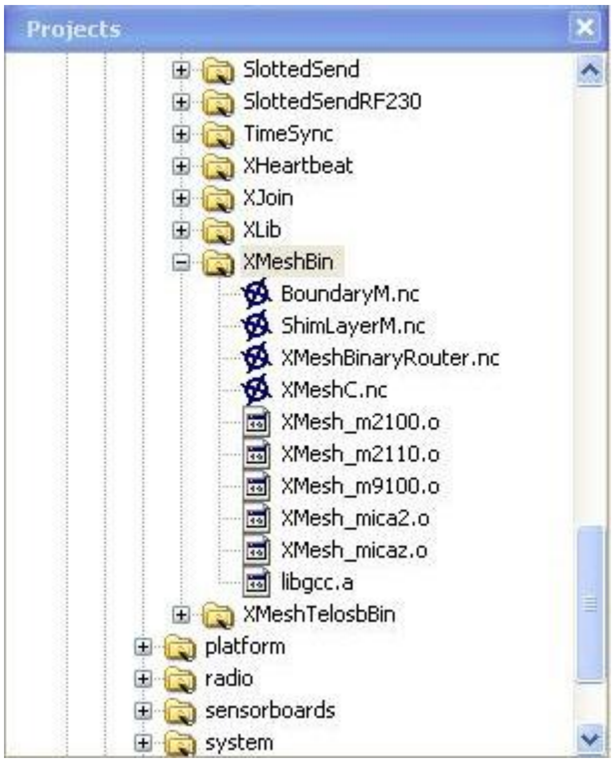


Figure 27 XMeshBin Directory

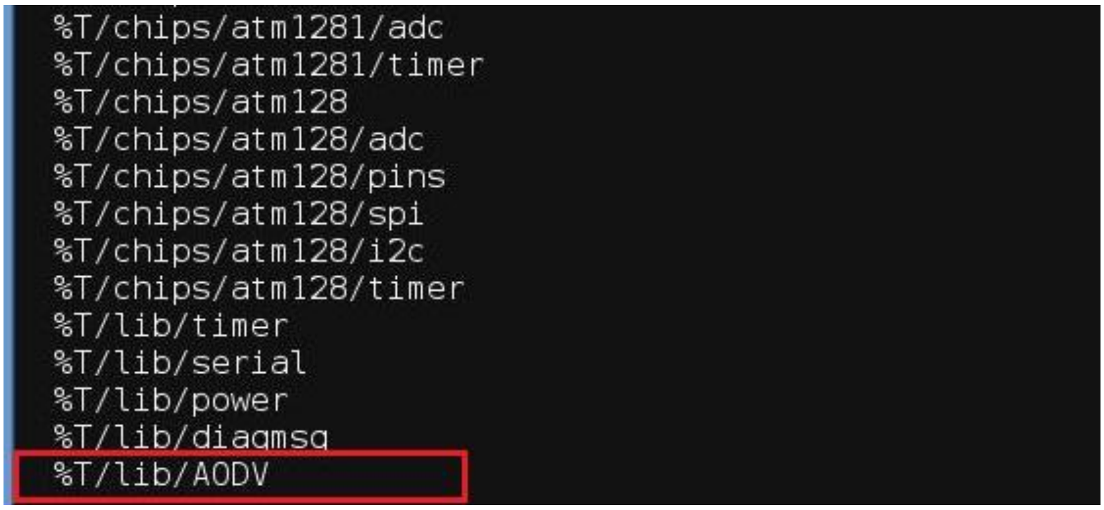
In the XMeshBin directory in Figure 27, we can see that the network layer of XMesh is controlled by this binary (compiled) file.

4.3 Prototype

As the previous concern matters, XMesh network routing protocol cannot be altered from the given manufacturer programming environment. The author decides to use another alternative as purchasing the Enterprise Edition of the source code will be cost consuming and the possibilities of getting one is low as Crossbow Inc. no longer support the XMesh, MEMSIC does. The author has also written an email to the MEMSIC about the matters, but no reply given.

4.3.1 Ad-Hoc On-Demand Vector(AODV) Routing Implementation

The alternative method decided by the author is to implement AODV routing in the legacy TinyOS-2.x programming environment. Installing TinyOS-2.x in Linux (XUbuntu) will provide the programming environment. The author has also tried using TinyOS in Cygwin but some problems arise concerning Java-TinyOS compilation. The method used in installing TinyOS-2.x in Linux has been described in SECTION 3.6.4.



```
%T/chips/atm1281/adc
%T/chips/atm1281/timer
%T/chips/atm128
%T/chips/atm128/adc
%T/chips/atm128/pins
%T/chips/atm128/spi
%T/chips/atm128/i2c
%T/chips/atm128/timer
%T/lib/timer
%T/lib/serial
%T/lib/power
%T/lib/diagmsg
%T/lib/AODV
```

Figure 28 AODV's ".platform" modification

AODV implementation and documentation can be obtained from a documentation website of a PhD Candidate, [**17**]. Its source code was published under the GNU License.

After obtaining the source code, it then can be copied to $TOSROOT/tos and $TOSROOT/apps. Some adjustments need to be done before being able to install the sample application on a mote, the adjustment is described below.

1. Go to *$TOSROOT/tos/platform/<your platform>*. In the author's case, it is *$TOSROOT/tos/platform/iris.*
2. Edit the .platform file (the file may be hidden, editing the option of file manager will show the file)
3. Add a new line with the new code as in Figure 28.

The implementation continues by installing the given example of application layer's code named as AODV25nodeTest. Several adjustments in the code were done to make it working. There are several error inside the code, discovered by the author through Leds debugging. The cause of the error possibly from the written code itself or it is because of different version of TinyOS used, noted that the author used TinyOS-2.1.0. The adjustments are listed below:

1. In AODV_M.nc file, the use of function "add_route_table()" by the variable "added" in "ReceiveRREQ.receive()" event. The "added" need to be removed from IF statement as it will return boolean FALSE. The add_route_table need to be added inside the IF statement, not in the boolean operation. Refer to Appendix B.1.
2. In AODV_M.nc file, the "p=msg" should be removed as the event need to be returned with message_t* format. Signalling the event alone is enough. If this correction not been done, the mote will actually receive the sent packet and data, but the function SubReceive.receive will be corrupted after some time. The author discovered this by watching using Leds debug that the mote only receive data in upper layer for 2-3 packet only. Refer to appendix B.2.

The motes then are able to send data to each other after the above corrections.

4.3.2 Adding Battery's Factor in the AODV

The idea is to add a certain factor of battery's remaining power inside the AODV routing algorithm. The justification of this is the forwarding mote that has less remaining battery's power will be able to hold longer thus being able to send its own sensor data to the base station.

It is advisable to read the documentation of AODV Routing Algorithm [**17**] to understand the following explanation. This routing technique will allow overall motes in a network to stay alive as well as decreasing the rate of battery replacement due to motes that are in the middle and forwarding a lot of packets. The user will mostly receive all the critical sensor data in a control room.

The battery's power of a mote will be sensed for every certain period and the request of route will be added with Minimum remaining Battery in a Transmission (MBT). The explanation can be simplified with Figure 29.

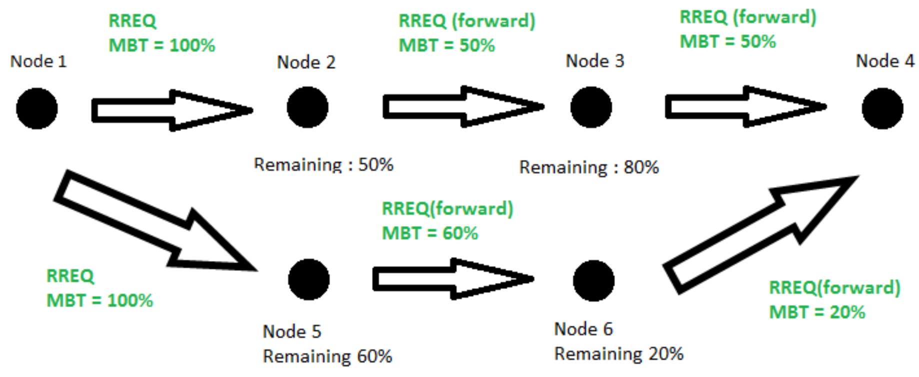


Figure 29 Minimum remaining Battery in a Transmission

The example scenario describe that Node 1 need to transmit data to Node 4 and there is no route path yet in its routing table. Node 1 will transmit RouteRequest(RREQ) with 100% MBT as the sender node should not affect the MBT as the node need to receive data packet nevertheless. The RREQ will be broadcast to all neighbour motes. Node 4 then will reply all request back to Node 1. Node 1 will then filter the RREP, which one has the highest MBT to be stored in its routing table. Please note that the situation should has more route path (e.x. Node 1-2-6-4), but it is excluded in the Figure 29.

This routing technique has been implemented with AODV by the author. The following steps described the implementation on the source code, the network layer source code of AODV can be found in $TOSROOT/lib/ADOV :

1. Wire the needed components-interface in AODV.nc, another Timer and BatteryVoltage sensor. Declare the needed interface in AODV_M.nc.
2. The new Timer functions as to periodic the sensor reading of the battery's voltage. Start the Timer in AODV_M.nc.
3. When the Timer fired, the sensor will be read, the data is calculated and store in a local variable.
4. Update the structure of RREQ, RREP and Route Table.
5. For first RREQ send by the requester, the MBT should be 100%.
6. The RREQ that is being received and not for it (forwarding RREQ), the forwarder node should update the MBT as necessary. As well as updating its routing table with the received MBT from the last sender node.
7. During the route table updating function (add_route_table), the function should check whether the existing route path need to be replaced.
8. Change other required function for proper execution of code.
9. Refer to APPENDIX C for further clarification of code.

CHAPTER 5
RECOMMENDATION AND CONCLUSION

5.1 Conclusion

The project is running smoothly and within the time scope and schedule. The basic working network has been implemented though Moteworks. The MATLAB interface can be used for further process of data received from the node as it provides flexibility. Ad-Hoc On Demand Vector Routing has been implemented on IRIS motes with some adjustment from GNU Licensed code. The battery factor modification in AODV has been implemented, but during this report writing, it has not been thoroughly tested for comparison with other routing algorithm.

5.1 Recommendation

The development of WSN and Mesh networking need to done in a Linux environment such as Xubuntos that has been compiled for TinyOS development. The usage of Cygwin and Windows has many disadvantages and problems.

Further studies should be started with TinyOS, not some other routing algorihtm given by any manufacturer for better understanding and flexibility. Altering the code and library will be the further steps in improving the system dynamically.

REFERENCES

[1] P. Clare Loren, J. Pottie Gregory, and R. Agre Jonathan, "Self-Organizing Distributed Sensor Networks," in *Proc. SPIE Aerosense99*, CA, 1999.

[2] Microsoft. (2009) msn. [Online]. encarta.msn.com

[3] F. Akyildiz Ian and Wang Xudang, "A Survey on Wireless Mesh Networks," in *IEEE Radio Communications*, 2005, p. S23.

[4] Teo Cheng Kiat Amos, "Performance Evaluation of a Routing Protocol in Wireless Sensor Network," Naval Postgradute School, Monterey, CA, Master's Thesis NSN 7540-01-280-5500, 2005.

[5] W. Davis Tyler, Liang Xu, Navarro Miguel, Bhatnagar Diviyansh, and Liang Yao, "An Experimental Study of WSN Power Effieciency: MICAz Networks with XMesh," *International Journal of Distributed Sensor Networks*, vol. 2012, October 2011.

[6] Akyildiz LF., Su W, Sankarasubramaniam Y., and Cayirci E, "Wireless Sensor Networks: A Survey," *Communication Magazine*, vol. 40, no. 8, pp. 102-114, August 2002.

[7] V. Biradar Rajashree, Patil V.C., Sawant Dr. S. R., and Mudholkar Dr. R. R., "Classification and Comparison of Routing Protocols in Wireless Sensor Networks," *Ubiquitous Computing Security Systems*, no. Special, August 2009.

[8] MEMSIC Inc., *Xmesh User Manual*. Milpitas: MEMSIC, Inc., 2010.

[9] Baleri Giri, *Guidelines for WSN Design and Deployment*.: Crossbow Technology, Inc.

[10] Yajie Ma, Mark Richards, Moustafa Ghanem, Yike Guo, and John Hassard, "Air Pollution Monitoring and Mining Based on Sensor Grid in London," *Sensors for Urban Environmental Monitoring*, vol. 8, no. Special, 2008.

[11] Dust Networks. [Online]. http://www.dustnetworks.com/technology/networkintelligence/

[12] Nageswara Rao S. Siva, Krishna Y.K Sundara, and Rao K. Nageswara, "A Surve: Routing Protocols for Wireless Mesh Networks," *International Journal of Research and Reviews in Wireless Sensor Networks (IJRRWSN)*, vol. 1, no. 3, pp. 43-47, September 2011.

[13] MEMSIC, Inc., *Moteworks User Manual*.: MEMSIC, Inc., 2010.

[14] Kevin Klues. [TinyOS Wiki] Running a XubunTOS Virtual Machine Image in VMware Player. [Online]. http://docs.tinyos.net/tinywiki/index.php/Running_a_XubunTOS_Virtual_Machine_Image_in_VMware_Player

[15] Amos Teo, Gurminder Singh, and John C. McFaclien, "Evaluation of the XMesh Routing Protocol in Wireless Sensor Networks," 2006.

[16] Giri Baleri. (2007) [Tinyos-help] missing XMeshRouter file? [Online]. http://mail.millennium.berkeley.edu/pipermail/tinyos-help/2007-June/026203.html

[17] Junseok Kim. (2011, April) AODV implementation on TinyOS-2.x. [Online]. http://www2.engr.arizona.edu/~junseok/AODV.html

[18] Krishnendu Chakrabarty Yi Zou, "Sensor deployment and target localization in distributed sensor networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 1, pp. 61-91, February 2004.

[19] Chi-Tsun Cheng. (2011, September) tdoa-localization-iris, GoogleCode. [Online]. http://tdoa-localization-iris.googlecode.com/files/TinyOS%202x%20Installation%20Procedures.pdf

APPENDIX

APPENDIX A.1

MATLAB-M files :

```
s = dos('base_read.bat &');

keeplooping = true;

while keeplooping

conn =
database('task','tele','tiny','org.postgresql.Driver','jdbc:postgre
sql://localhost:5432/');

setdbprefs('DataReturnFormat','numeric')

xnodeid = 1;

nodeid = num2str(xnodeid);

xday = 15;

if xday < 10

    day = ['0',num2str(xday)];

else

    day = num2str(xday);

end


xmonth = 10;

if xmonth < 10

    month = ['0',num2str(xmonth)];

else

    month = num2str(xmonth);

end
```

```
xyear = 2012;

year = num2str(xyear);

curs = exec(conn, ['SELECT EXTRACT(HOUR FROM
result_time),EXTRACT(MINUTE FROM result_time),EXTRACT(SECOND FROM
result_time),humid,accel_x,accel_y FROM mts420_results WHERE
EXTRACT(DAY FROM result_time)=''', day,'''','AND EXTRACT(MONTH FROM
result_time)=''', month,'''','AND EXTRACT(YEAR FROM
result_time)=''', year,'''','AND nodeid=''', nodeid,'''']);

curs = fetch(curs);

hour = curs.Data(:,1);

minutes = curs.Data(:,2);

seconds = curs.Data(:,3);

humid = curs.Data(:,4);

accel_x = curs.Data(:,5);

accel_y = curs.Data(:,6);


n = length(humid);

nyear = (xyear * ones(1,n))';

nmonth = (xmonth * ones(1,n))';

nday = (xday * ones(1,n))';


xdate = datenum(nyear,nmonth,nday,hour,minutes,seconds);

%plot(humid)

subplot(2,2,[1 3]); plot(xdate,humid)

title('Humidity')

datetick('x','HH:MM:SS','keepticks')

subplot(2,2,2); plot(xdate,accel_x)

title('X-acceleration')

datetick('x','HH:MM:SS','keepticks')

subplot(2,2,4); plot(xdate,accel_y)
```

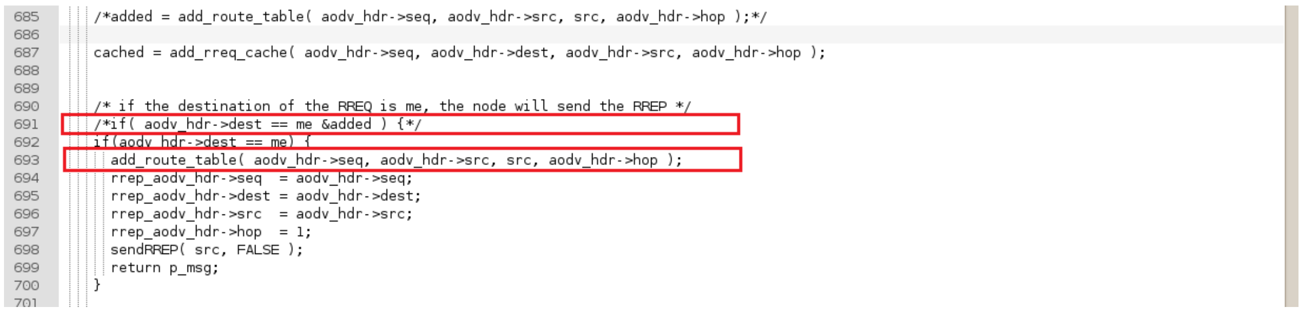
```
title('Y-acceleration')

datetick('x','HH:MM:SS','keepticks')


pause(1);

end

close(curs)

close(conn)
```

APPENDIX A.2

BATCH file (used by MATLAB to run Xserve)

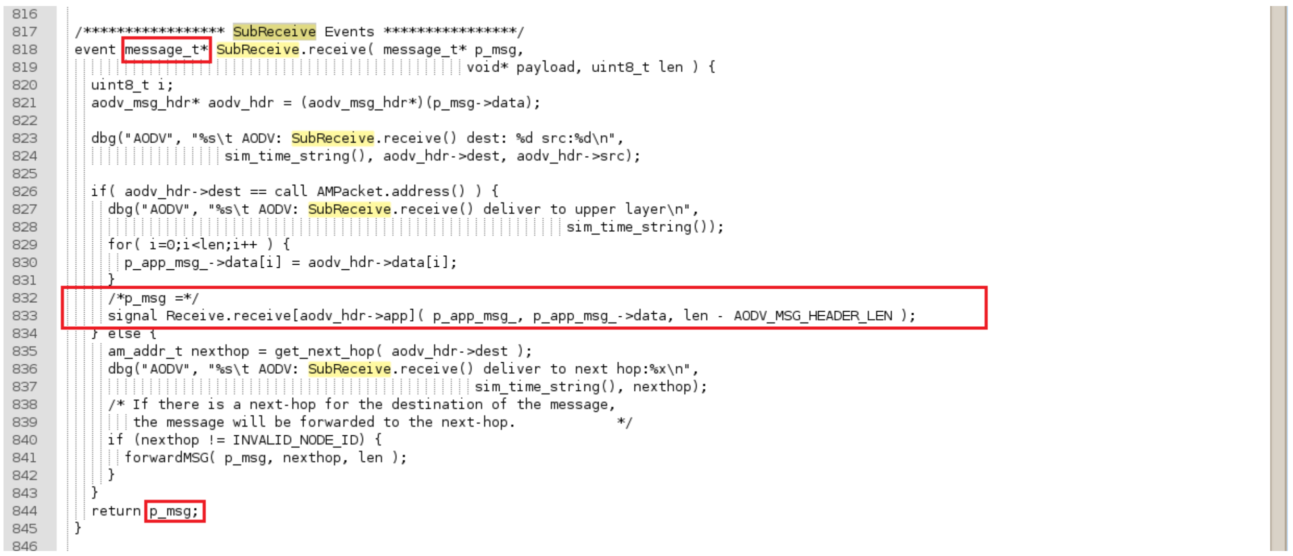
```
@echo off

echo off

c:

cd\

cd Program Files\Crossbow\MoteView\xserve2\bin

xserve -s=COM4 -b=57600 -l -xmlfile=XmlStream.xml -xmlp -
xmlport=9005
```

APPENDIX B.1



```
/*added = add_route_table( aodv_hdr->seq, aodv_hdr->src, src, aodv_hdr->hop );*/

cached = add_rreq_cache( aodv_hdr->seq, aodv_hdr->dest, aodv_hdr->src, aodv_hdr->hop );


/* if the destination of the RREQ is me, the node will send the RREP */
/*if( aodv_hdr->dest == me &added ) {*/
if(aodv_hdr->dest == me) {
  add_route_table( aodv_hdr->seq, aodv_hdr->src, src, aodv_hdr->hop );
  rrep_aodv_hdr->seq  = aodv_hdr->seq;
  rrep_aodv_hdr->dest = aodv_hdr->dest;
  rrep_aodv_hdr->src  = aodv_hdr->src;
  rrep_aodv_hdr->hop  = 1;
  sendRREP( src, FALSE );
  return p_msg;
}
```

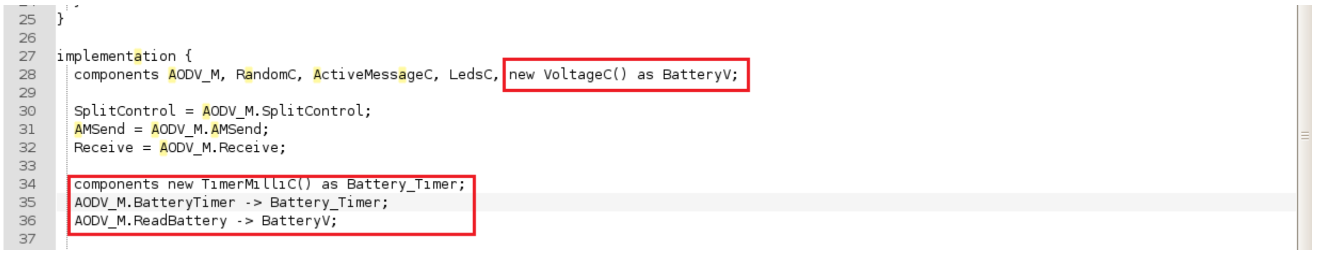
APPENDIX B.2



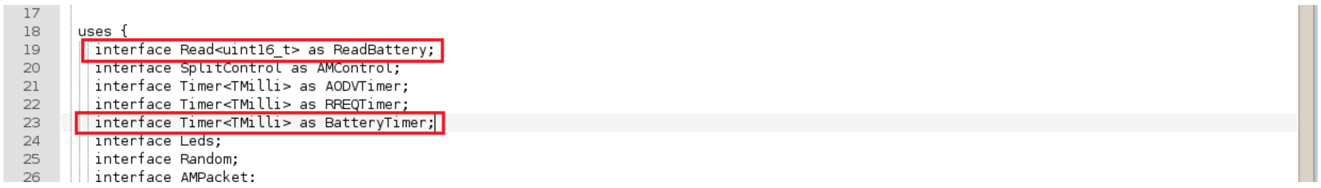
```

/***************** SubReceive Events ****************/
event message_t* SubReceive.receive( message_t* p_msg,
                                     void* payload, uint8_t len ) {
  uint8_t i;
  aodv_msg_hdr* aodv_hdr = (aodv_msg_hdr*)(p_msg->data);

  dbg("AODV", "%s\t AODV: SubReceive.receive() dest: %d src:%d\n",
                sim_time_string(), aodv_hdr->dest, aodv_hdr->src);

  if( aodv_hdr->dest == call AMPacket.address() ) {
    dbg("AODV", "%s\t AODV: SubReceive.receive() deliver to upper layer\n",
                                                 sim_time_string());
    for( i=0;i<len;i++ ) {
      p_app_msg_->data[i] = aodv_hdr->data[i];
    }
    /*p_msg =*/
    signal Receive.receive[aodv_hdr->app]( p_app_msg_, p_app_msg_->data, len - AODV_MSG_HEADER_LEN );
  } else {
    am_addr_t nexthop = get_next_hop( aodv_hdr->dest );
    dbg("AODV", "%s\t AODV: SubReceive.receive() deliver to next hop:%x\n",
                                          sim_time_string(), nexthop);
    /* If there is a next-hop for the destination of the message,
       the message will be forwarded to the next-hop.            */
    if (nexthop != INVALID_NODE_ID) {
      forwardMSG( p_msg, nexthop, len );
    }
  }
  return p_msg;
}
```

APPENDIX C

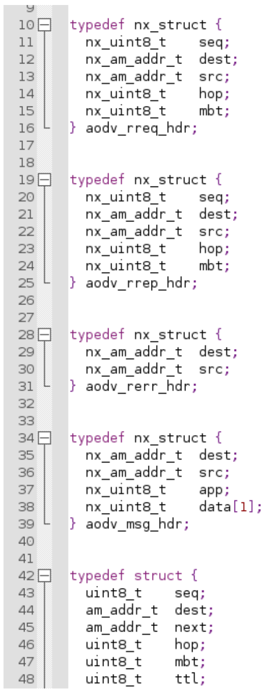
AODV.nc wiring modified code:


```
}

implementation {
  components AODV_M, RandomC, ActiveMessageC, LedsC, new VoltageC() as BatteryV;

  SplitControl = AODV_M.SplitControl;
  AMSend = AODV_M.AMSend;
  Receive = AODV_M.Receive;

  components new TimerMilliC() as Battery_Timer;
  AODV_M.BatteryTimer -> Battery_Timer;
  AODV_M.ReadBattery -> BatteryV;

```

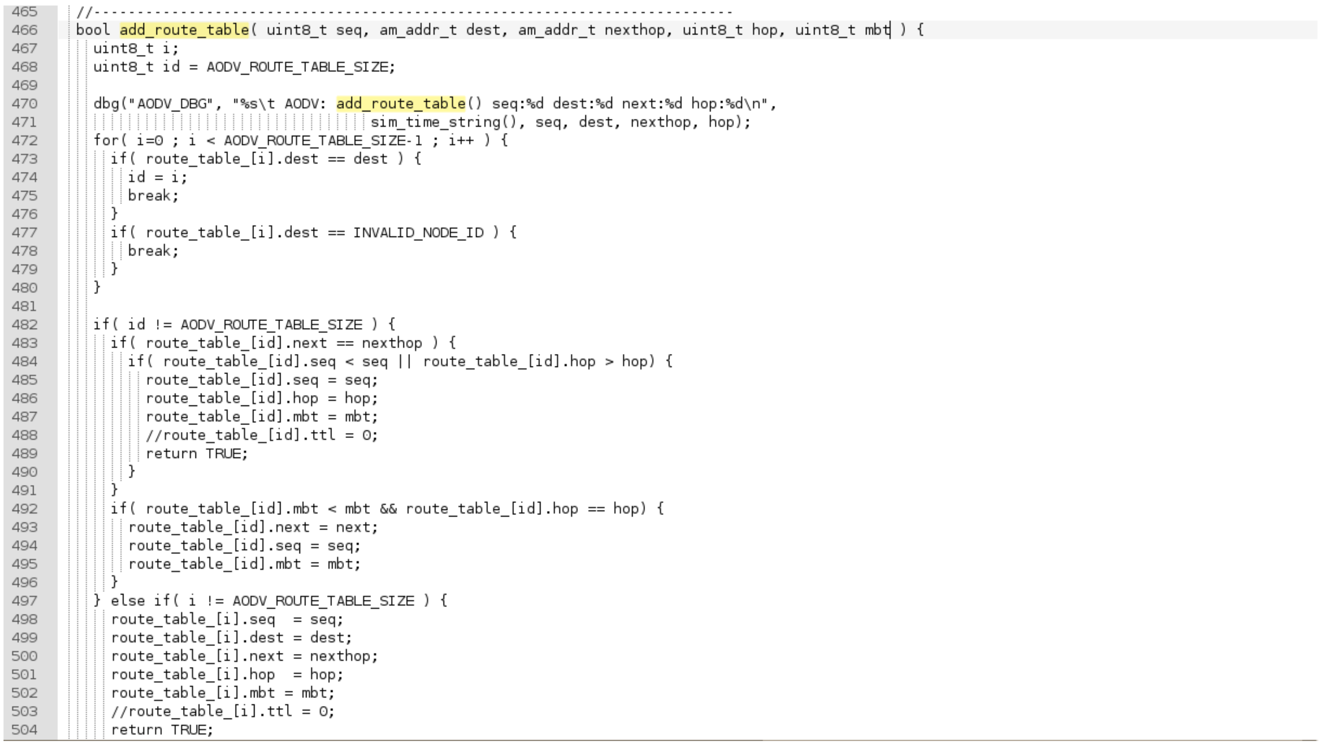
AODV_M.nc interface wiring modified code:


```

  uses {
    interface Read<uint16_t> as ReadBattery;
    interface SplitControl as AMControl;
    interface Timer<TMilli> as AODVTimer;
    interface Timer<TMilli> as RREQTimer;
    interface Timer<TMilli> as BatteryTimer;
    interface Leds;
    interface Random;
    interface AMPacket:
```

AODV.h modified code:

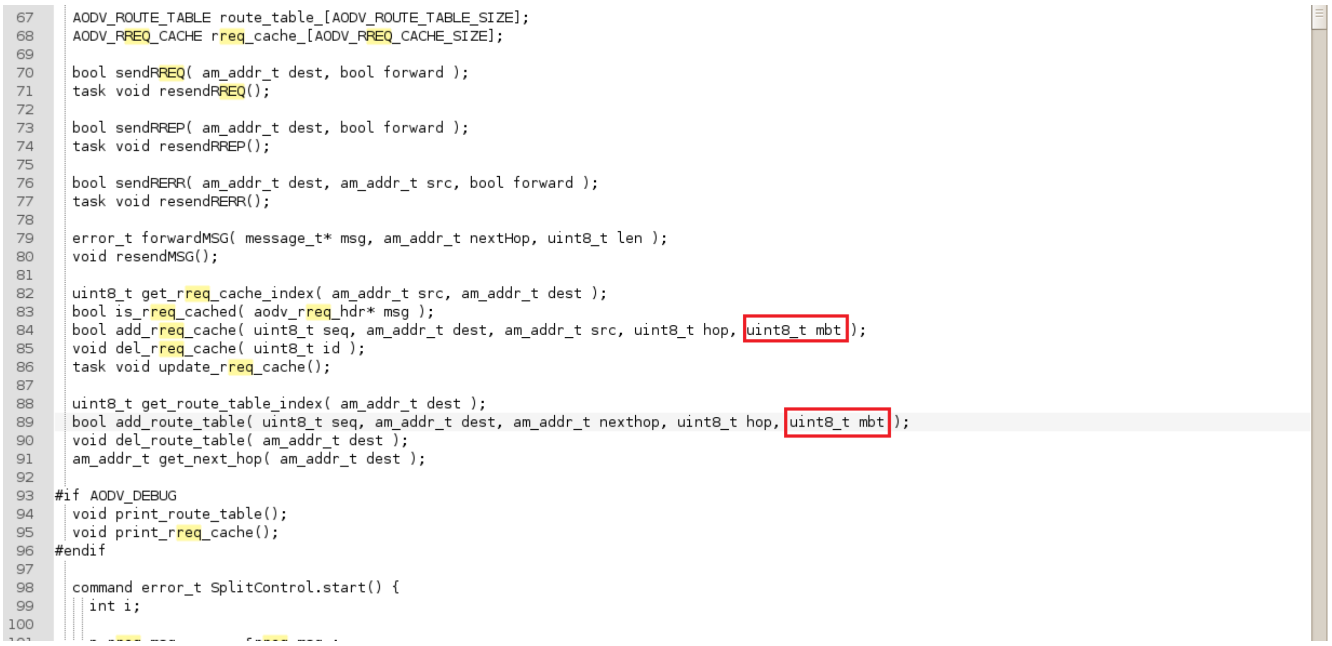

```
typedef nx_struct {
  nx_uint8_t    seq;
  nx_am_addr_t  dest;
  nx_am_addr_t  src;
  nx_uint8_t    hop;
  nx_uint8_t    mbt;
} aodv_rreq_hdr;


typedef nx_struct {
  nx_uint8_t    seq;
  nx_am_addr_t  dest;
  nx_am_addr_t  src;
  nx_uint8_t    hop;
  nx_uint8_t    mbt;
} aodv_rrep_hdr;


typedef nx_struct {
  nx_am_addr_t  dest;
  nx_am_addr_t  src;
} aodv_rerr_hdr;


typedef nx_struct {
  nx_am_addr_t  dest;
  nx_am_addr_t  src;
  nx_uint8_t    app;
  nx_uint8_t    data[1];
} aodv_msg_hdr;


typedef struct {
  uint8_t    seq;
  am_addr_t  dest;
  am_addr_t  next;
  uint8_t    hop;
  uint8_t    mbt;
  uint8_t    ttl;
```

Add_route_table function in AODV_M.nc :



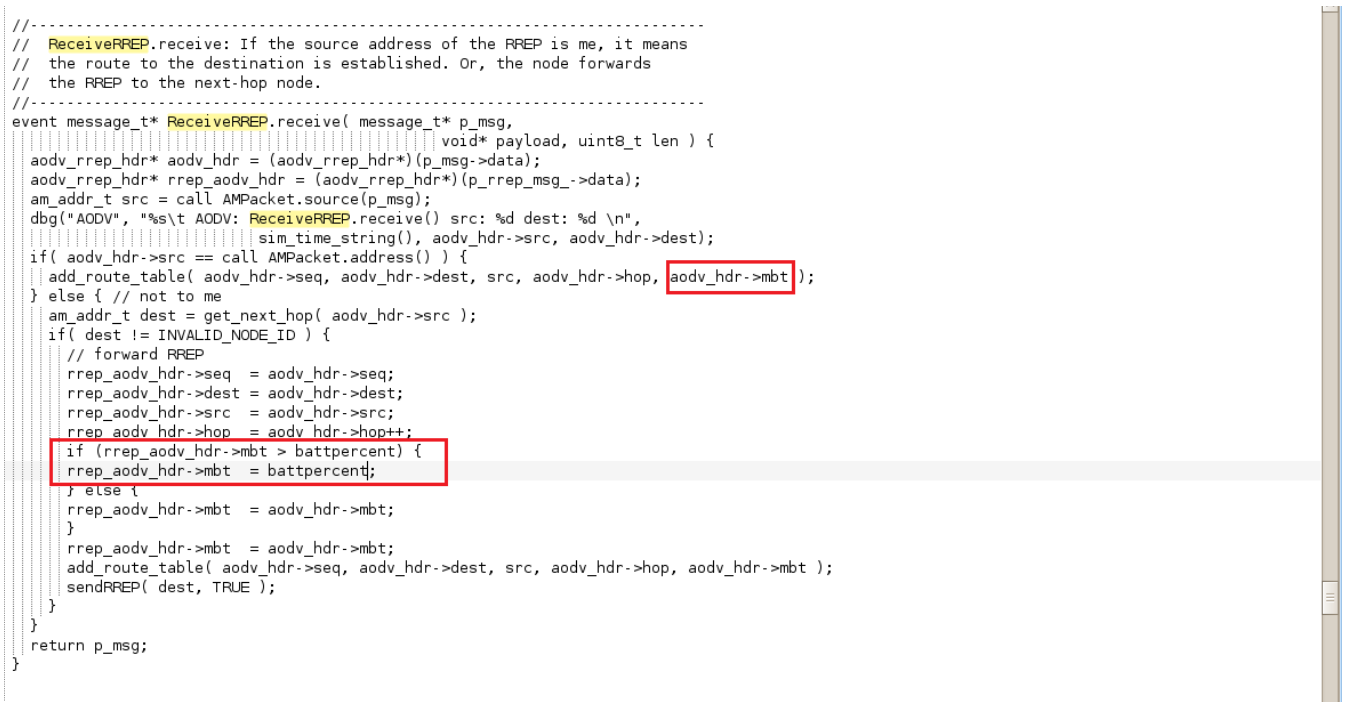
```
//--------------------------------------------------------------------------
bool add_route_table( uint8_t seq, am_addr_t dest, am_addr_t nexthop, uint8_t hop, uint8_t mbt ) {
  uint8_t i;
  uint8_t id = AODV_ROUTE_TABLE_SIZE;

  dbg("AODV_DBG", "%s\t AODV: add_route_table() seq:%d dest:%d next:%d hop:%d\n",
                                  sim_time_string(), seq, dest, nexthop, hop);
  for( i=0 ; i < AODV_ROUTE_TABLE_SIZE-1 ; i++ ) {
    if( route_table_[i].dest == dest ) {
      id = i;
      break;
    }
    if( route_table_[i].dest == INVALID_NODE_ID ) {
      break;
    }
  }

  if( id != AODV_ROUTE_TABLE_SIZE ) {
    if( route_table_[id].next == nexthop ) {
      if( route_table_[id].seq < seq || route_table_[id].hop > hop) {
        route_table_[id].seq = seq;
        route_table_[id].hop = hop;
        route_table_[id].mbt = mbt;
        //route_table_[id].ttl = 0;
        return TRUE;
      }
    }
    if( route_table_[id].mbt < mbt && route_table_[id].hop == hop) {
      route_table_[id].next = next;
      route_table_[id].seq = seq;
      route_table_[id].mbt = mbt;
    }
  } else if( i != AODV_ROUTE_TABLE_SIZE ) {
    route_table_[i].seq  = seq;
    route_table_[i].dest = dest;
    route_table_[i].next = nexthop;
    route_table_[i].hop  = hop;
    route_table_[i].mbt = mbt;
    //route_table_[i].ttl = 0;
    return TRUE;
```

Function declaration in AODV_M.nc :



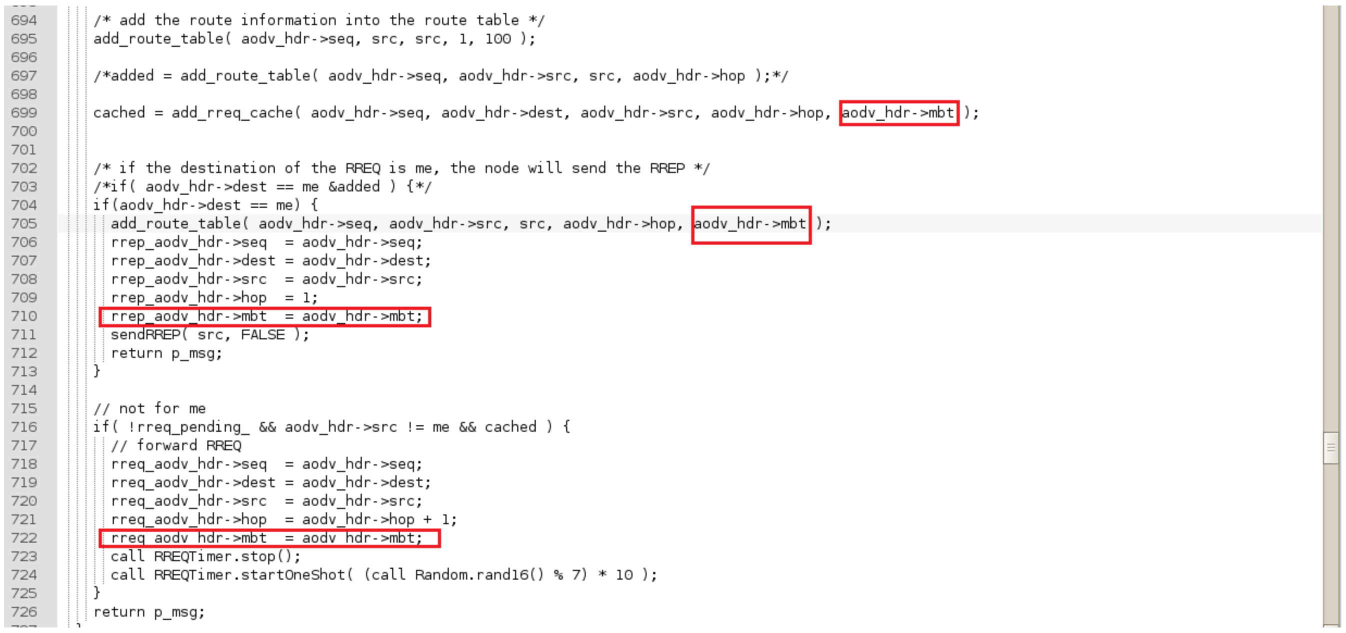
```
  AODV_ROUTE_TABLE route_table_[AODV_ROUTE_TABLE_SIZE];
  AODV_RREQ_CACHE rreq_cache_[AODV_RREQ_CACHE_SIZE];

  bool sendRREQ( am_addr_t dest, bool forward );
  task void resendRREQ();

  bool sendRREP( am_addr_t dest, bool forward );
  task void resendRREP();

  bool sendRERR( am_addr_t dest, am_addr_t src, bool forward );
  task void resendRERR();

  error_t forwardMSG( message_t* msg, am_addr_t nextHop, uint8_t len );
  void resendMSG();

  uint8_t get_rreq_cache_index( am_addr_t src, am_addr_t dest );
  bool is_rreq_cached( aodv_rreq_hdr* msg );
  bool add_rreq_cache( uint8_t seq, am_addr_t dest, am_addr_t src, uint8_t hop, uint8_t mbt );
  void del_rreq_cache( uint8_t id );
  task void update_rreq_cache();

  uint8_t get_route_table_index( am_addr_t dest );
  bool add_route_table( uint8_t seq, am_addr_t dest, am_addr_t nexthop, uint8_t hop, uint8_t mbt );
  void del_route_table( am_addr_t dest );
  am_addr_t get_next_hop( am_addr_t dest );

#if AODV_DEBUG
  void print_route_table();
  void print_rreq_cache();
#endif

  command error_t SplitControl.start() {
    int i;

```

ReceiveRREP function in AODV_M.nc :



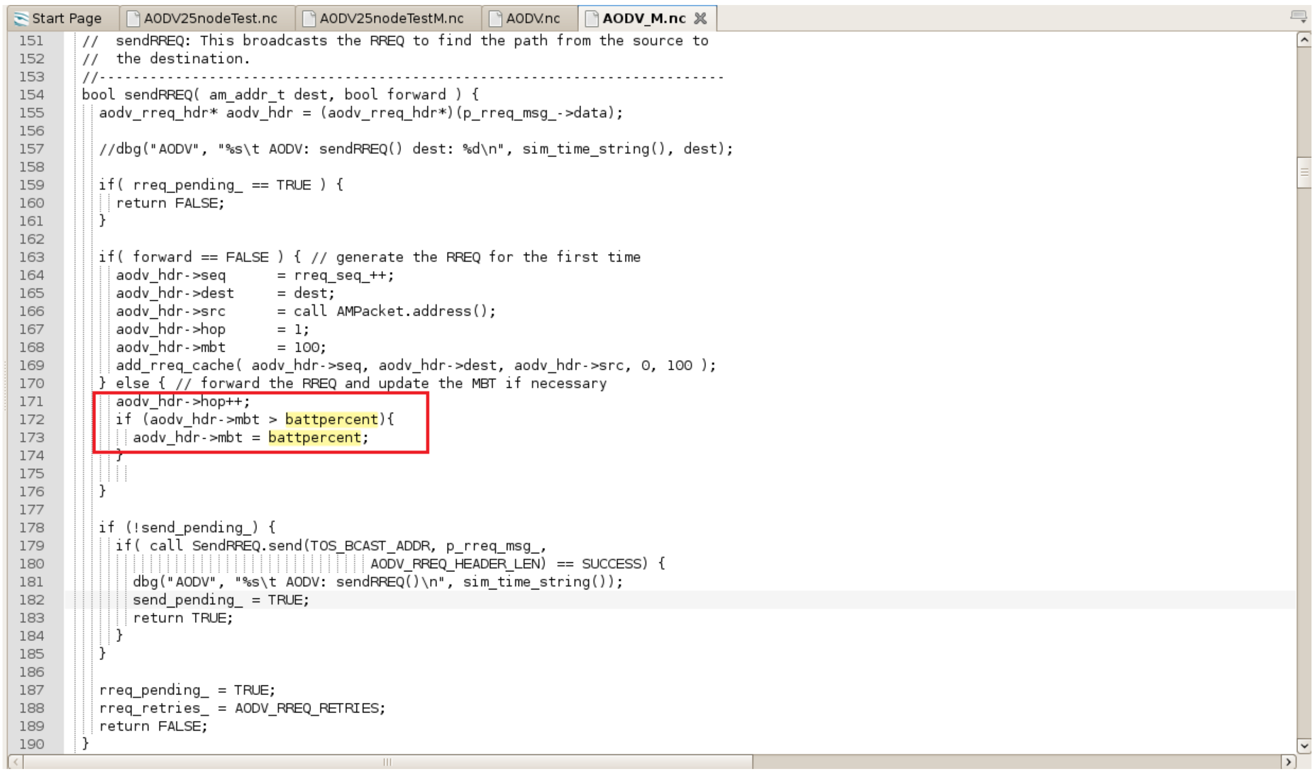
```
//------------------------------------------------------------------------
//  ReceiveRREP.receive: If the source address of the RREP is me, it means
//  the route to the destination is established. Or, the node forwards
//  the RREP to the next-hop node.
//------------------------------------------------------------------------
event message_t* ReceiveRREP.receive( message_t* p_msg,
                                               void* payload, uint8_t len ) {
  aodv_rrep_hdr* aodv_hdr = (aodv_rrep_hdr*)(p_msg->data);
  aodv_rrep_hdr* rrep_aodv_hdr = (aodv_rrep_hdr*)(p_rrep_msg_->data);
  am_addr_t src = call AMPacket.source(p_msg);
  dbg("AODV", "%s\t AODV: ReceiveRREP.receive() src: %d dest: %d \n",
                         sim_time_string(), aodv_hdr->src, aodv_hdr->dest);
  if( aodv_hdr->src == call AMPacket.address() ) {
    add_route_table( aodv_hdr->seq, aodv_hdr->dest, src, aodv_hdr->hop, aodv_hdr->mbt );
  } else { // not to me
    am_addr_t dest = get_next_hop( aodv_hdr->src );
    if( dest != INVALID_NODE_ID ) {
      // forward RREP
      rrep_aodv_hdr->seq  = aodv_hdr->seq;
      rrep_aodv_hdr->dest = aodv_hdr->dest;
      rrep_aodv_hdr->src  = aodv_hdr->src;
      rrep_aodv_hdr->hop  = aodv_hdr->hop++;
      if (rrep_aodv_hdr->mbt > battpercent) {
      rrep_aodv_hdr->mbt  = battpercent;
      } else {
      rrep_aodv_hdr->mbt  = aodv_hdr->mbt;
      }
      rrep_aodv_hdr->mbt  = aodv_hdr->mbt;
      add_route_table( aodv_hdr->seq, aodv_hdr->dest, src, aodv_hdr->hop, aodv_hdr->mbt );
      sendRREP( dest, TRUE );
    }
  }
  return p_msg;
}
```

ReceiveRREQ function in AODV_M.nc



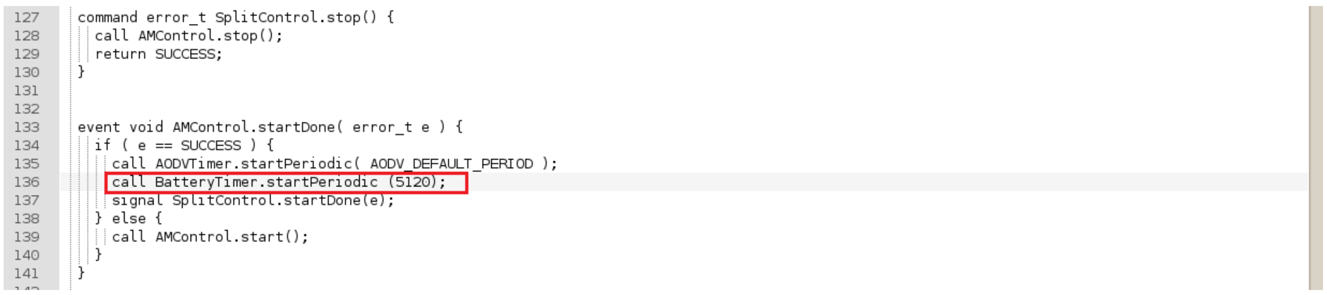
```
      /* add the route information into the route table */
      add_route_table( aodv_hdr->seq, src, src, 1, 100 );

      /*added = add_route_table( aodv_hdr->seq, aodv_hdr->src, src, aodv_hdr->hop );*/

      cached = add_rreq_cache( aodv_hdr->seq, aodv_hdr->dest, aodv_hdr->src, aodv_hdr->hop, aodv_hdr->mbt );


      /* if the destination of the RREQ is me, the node will send the RREP */
      /*if( aodv_hdr->dest == me &added ) {*/
      if(aodv_hdr->dest == me) {
        add_route_table( aodv_hdr->seq, aodv_hdr->src, src, aodv_hdr->hop, aodv_hdr->mbt );
        rrep_aodv_hdr->seq  = aodv_hdr->seq;
        rrep_aodv_hdr->dest = aodv_hdr->dest;
        rrep_aodv_hdr->src  = aodv_hdr->src;
        rrep_aodv_hdr->hop  = 1;
        rrep_aodv_hdr->mbt  = aodv_hdr->mbt;
        sendRREP( src, FALSE );
        return p_msg;
      }

      // not for me
      if( !rreq_pending_ && aodv_hdr->src != me && cached ) {
        // forward RREQ
        rreq_aodv_hdr->seq  = aodv_hdr->seq;
        rreq_aodv_hdr->dest = aodv_hdr->dest;
        rreq_aodv_hdr->src  = aodv_hdr->src;
        rreq_aodv_hdr->hop  = aodv_hdr->hop + 1;
        rreq_aodv_hdr->mbt  = aodv_hdr->mbt;
        call RREQTimer.stop();
        call RREQTimer.startOneShot( (call Random.rand16() % 7) * 10 );
      }
      return p_msg;
```

SendRREQ function in AODV_M.nc :

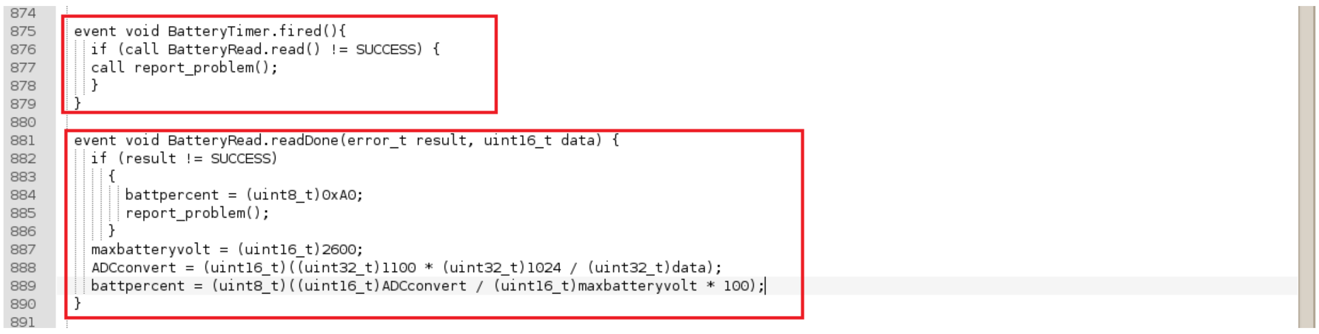


```
//  sendRREQ: This broadcasts the RREQ to find the path from the source to
//  the destination.
//------------------------------------------------------------------------
bool sendRREQ( am_addr_t dest, bool forward ) {
  aodv_rreq_hdr* aodv_hdr = (aodv_rreq_hdr*)(p_rreq_msg_->data);

  //dbg("AODV", "%s\t AODV: sendRREQ() dest: %d\n", sim_time_string(), dest);

  if( rreq_pending_ == TRUE ) {
    return FALSE;
  }

  if( forward == FALSE ) { // generate the RREQ for the first time
    aodv_hdr->seq      = rreq_seq_++;
    aodv_hdr->dest     = dest;
    aodv_hdr->src      = call AMPacket.address();
    aodv_hdr->hop      = 1;
    aodv_hdr->mbt      = 100;
    add_rreq_cache( aodv_hdr->seq, aodv_hdr->dest, aodv_hdr->src, 0, 100 );
  } else { // forward the RREQ and update the MBT if necessary
    aodv_hdr->hop++;
    if (aodv_hdr->mbt > battpercent){
      aodv_hdr->mbt = battpercent;
    }

  }

  if (!send_pending_) {
    if( call SendRREQ.send(TOS_BCAST_ADDR, p_rreq_msg_,
                              AODV_RREQ_HEADER_LEN) == SUCCESS) {
      dbg("AODV", "%s\t AODV: sendRREQ()\n", sim_time_string());
      send_pending_ = TRUE;
      return TRUE;
    }
  }

  rreq_pending_ = TRUE;
  rreq_retries_ = AODV_RREQ_RETRIES;
  return FALSE;
}
```

Timer in AODV_M.nc :



```
command error_t SplitControl.stop() {
  call AMControl.stop();
  return SUCCESS;
}


event void AMControl.startDone( error_t e ) {
  if ( e == SUCCESS ) {
    call AODVTimer.startPeriodic( AODV_DEFAULT_PERIOD );
    call BatteryTimer.startPeriodic (5120);
    signal SplitControl.startDone(e);
  } else {
    call AMControl.start();
  }
}
```



```
event void BatteryTimer.fired(){
  if (call BatteryRead.read() != SUCCESS) {
  call report_problem();
  }
}

event void BatteryRead.readDone(error_t result, uint16_t data) {
  if (result != SUCCESS)
    {
      battpercent = (uint8_t)0xA0;
      report_problem();
    }
  maxbatteryvolt = (uint16_t)2600;
  ADCconvert = (uint16_t)((uint32_t)1100 * (uint32_t)1024 / (uint32_t)data);
  battpercent = (uint8_t)((uint16_t)ADCconvert / (uint16_t)maxbatteryvolt * 100);
}
```