RIMBAMON© : A Forest Monitoring System Using Wireless Sensor Networks

By

MUHAMAD HAIDAR BIN SUHAIMI

FINAL PROJECT REPORT

Submitted to the Electrical & Electronics Engineering Programme
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Universiti Teknologi Petronas
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

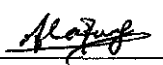
RIMBAMON© : A Forest Monitoring System Using Wireless Sensor Networks

by

Muhamad Haidar Bin Suhaimi

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Approved:



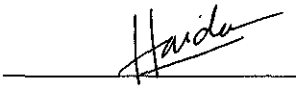
Mr. Azlan Awang
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

JUNE 2007

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



Muhamad Haidar Bin Suhaimi

ABSTRACT

Forests are important to the lives of every human and animal living on this planet. It provides an environment for many species of plants and animals, thus protecting and sustaining the diversity of nature. However, it is constantly being threatened by illegal logging and indiscriminate development. This problem needs to be overcome in order to protect our forests before it is too late. Therefore, there is an urgent need to constantly monitor the conditions inside the forest especially under the forest canopy in order to develop a comprehensive, real-time Forest Decision Support System. This report presents the RIMBAMON© system which is aimed at providing a way to capture and manipulate forestry environmental data coming from the sensors in a sensor network. This system utilizes Wireless Sensor Networks for communication. As there are several existing technologies that employ the Wireless Sensor Networks technology, this project will focus on the TinyOS operating system for sensor networks. Various tools will be used to develop the system; such as the TinyOS Simulator (TOSSIM), Java, Cygwin and TinyViz. The concepts behind wireless sensor networks and the tools used in this project are examined and discussed alongside simulation and testing of TinyOS. The outcome of this project is a prototype system which is able to monitor and report the conditions under the forest canopy.

ACKNOWLEDGEMENTS

First of all, my gratitude goes to Allah S.W.T for giving me strength, knowledge and good health during the course of my Final Year Project.

My warmest gratitude goes to my supervisor, Mr. Azlan Awang for his unwavering support and guidance while completing this project. His generosity, understanding and help have been an inspiration to me.

My special appreciation goes to Mr Wan Muzalif and Dr. Fadlee from Universiti Putra Malaysia for their valuable expertise, guide and support in completing this project.

Special thanks to my father, Dr. Suhaimi Napis, and mother, Dr. Faridah Noor, who gave their time and effort in guiding and assisting me throughout completing the project.

I would also like to thank Nurulizzatulshima, Al-Afiq and all the people who have been involved directly or indirectly in completing this project. The contributions you have made throughout the development of this project have been really valuable to me.

Last but not least, my deepest gratitude and special thanks goes to my family and friends for their support and knowledge that they shared with me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Background of Study

The pristine and green conditions of Malaysian forests have slowly deteriorated due to illegal logging and indiscriminate development. It has indeed become a serious problem that needs attention before it is too late. Therefore, there is an urgent need to constantly monitor the conditions inside the forest especially under the forest canopy in order to develop a comprehensive, real-time Forest Decision Support System. A relatively lower cost, faster to deploy and real-time monitoring system which offers capabilities of sensing the conditions and provide true data in real-time need to be implemented within (on-site) the target area. Wireless Sensor Technology can be applied to implement this at a relatively lower cost than aerial or satellite imaging techniques. Aerial or satellite imaging techniques can only passively monitor conditions using imaging extrapolation and data approximation based on models and incapable of monitoring and capturing true data in situ (on-site) and in real-time.

A Wireless Sensor Technology is based on communication using Wireless Sensor Networks (WSN) and consists of spatially distributed autonomous devices (sensors) to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at different locations and the data can be monitored and retrieved in real-time. Other software that supports the systems development include Network Embedded Systems C (nesC), which is a variant of the C-programming language and also TinyOS, an open-source, component-based operating system designed for wireless embedded sensor networks

The aim of the project is to design a system that monitors the conditions under the forest canopy and reports the results graphically in order to provide surveillance on illegal logging activities and forest fires. This system utilizes radio-based sensors to create a wireless sensor network for relaying sensor data. The monitoring station provides system information and real-time data reporting and provides data feeds that can be utilized by a Forest Decision Support System.

This project is not intended as a replacement system for remote sensing of forest areas, but rather aims at complimenting the existing methods for gathering information. Therefore, the project is aimed at being integrated into the field-based sampling operations. The information gathered will provide valuable data to a decision support system.

1.2 Problem Statement

The degradation of forests in Malaysia is a serious problem that needs to be overcome in order to preserve its natural forests. Therefore, there is a need to constantly monitor the conditions inside a forest in order to aid a decision support system. Among the main conditions that need to be monitored are the temperature levels, humidity levels, light intensity, and sound. Consequently, a relatively lower cost, faster to deploy and real-time monitoring system which offers capabilities of sensing these elements need to be implemented inside the target area. A Wireless Sensor Network is capable of doing this at a relatively cheaper price than aerial or satellite imaging techniques. This Wireless Sensor Network will consist of multiple sensor nodes which communicate by creating a mesh network between each other. The sensors reports the information gathered by relaying captured data to other sensors within the network. At the heart of this network, lies a base station or base computer that monitors and gathers the information that is passed around in the network. This base station will be connected to the monitoring station through a long range connection, either via satellite or wired means.

Due to the small program footprints of these sensors, the data that is sent and received is contained in a compact and hard to understand message format in order to minimize the size and conserve valuable battery life [3]. This information can be converted and manipulated by applying certain algorithms or formulas to decode the messages and properly tag it. Therefore, a program for the sensors to gather necessary data and convey it back to the base station needs to be developed. In order to read and understand the information, there is a need to display and convey this information in an understandable manner. Also, a proper program to store and present the data gathered in tables, graphs and images with a user friendly interface will have to be created. This is where RIMBAMON© comes in. This system will provide a way of monitoring forest areas which do not already have specialized monitoring systems for research purposes and the prevention of forest degradation.

1.3 Objective and Scope of Study

The main objective of this project is to create a system for the monitoring of forest areas, using Wireless Sensor Networks. In addition to the creation of the system, this project will also involve simulation of a sensor network in order to test the system's functionality. The system would be divided into two parts which are the programs operating the sensor nodes and the main processing program which captures the data and information coming from the sensor network motes connected to a base station, and manipulate the data for easy monitoring and recording. This information will then be displayed to the user in graphs or tables depending on the requirements of the user. However, the data that is sent by the sensor motes are in a raw format which is hard to understand. Therefore, in order to record and output the data, the system would need to implement formulas or algorithms to convert the raw data coming from the sensor motes into an understandable format (engineering units).

This project will cover the study of Wireless Sensor Networks (WSN) concepts, communication between sensors, data and information transfer within a wireless sensor network, the TinyOS operating system for sensor networks, the TOSSIM sensor network simulator for TinyOS, wireless sensor network simulation

development, and also information processing and manipulation. In addition to these, this project will also involve programming using nesC and Java, and also database development and access using SQL. The development of the system will involve program development for the displaying and processing of the data captured from the sensor network.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

This chapter will discuss the research work which has been done in order to achieve the objectives of this project. Among the topics discussed are Wireless Sensor Networks, the forest situation in Malaysia, current trends in forest remote sensing and the sensors used in this project.

2.2 Wireless Sensor Networks (WSN)

A wireless sensor network (WSN) is a computer network consisting of many, spatially distributed devices using sensors to monitor conditions at different locations, such as temperature, sound, vibration, pressure, motion or pollutants [1][2]. As these devices are small and inexpensive, they can be produced and deployed in large numbers. The size and price requirements imply that the devices' resources in terms of energy, memory, computational speed and bandwidth are severely constrained [3]. Each device is equipped with a radio transceiver, a small microcontroller, and an energy source, usually a battery. Each device relays information from other devices to transport data to a monitoring computer.

Although computer-based instrumentation has existed for a long time, the density of instrumentation made possible by a shift to produce mass-produced intelligent sensors and the use of pervasive networking technology gives wireless sensor networks a new kind of scope that can be applied to a wide range of users [6]. These users can be categorized into several groups:

- Space monitoring
- Object monitoring
- Object interaction and encompassing space monitoring

The first group includes environmental and habitat monitoring, indoor climate control, surveillance and intelligent alarms. The second includes structural monitoring, ecophysiology, condition-based equipment maintenance, medical diagnostics and urban terrain mapping. The most important applications involve monitoring complex interactions, including wildlife habitats, disaster management, emergency response, ubiquitous computing environments, asset tracking, healthcare and manufacturing process flow.

The organization of WSN consists of several management plan and infrastructural layers [4]. These plans and layers define the way the mesh network connections within a WSN area are established and the underlying technologies involved in WSN's.

The first management plan is that of power management. This plan concerns how the sensor node uses its stored power effectively and efficiently. The second management plan is mobility management which detects and registers the movement of sensor nodes or rather recognizes each neighboring sensor and balances their power and task usage. Lastly, there is the task management with the main aim and purpose of balancing and the sensing tasks given to a specific region of sensors. These plans work together to manage the Wireless Sensor Network and are interdependent with the infrastructure layers.

The infrastructure layers consist of five different layers, namely the Physical, Data-link, Network, Transport and Application layers [4]. The following is a brief definition of the functions of each layer:

(i) Physical Layer

Responsible for the frequency selection, transmission modulation and data encryption of messages sent.

(ii) Data-link Layer

Responsible for multiplexing the data streams, performs data frame detection, Medium Access (MAC) and error control.

(iii) Network Layer

Routes the data supplied by the Transport Layer through the use of special multi-hop wireless routing protocols between sensor nodes and sink/base station nodes.

(iv) Transport Layer

Maintains the flow of data if the Application Layer requires it. Required if there is a need for an end user to access the Sensor Network through the Internet.

(v) Application Layer

Makes the hardware and software of the lower layers transparent to the End-User.

2.3 Forest Situation in Malaysia

Tropical Forests are exceptional sources of biodiversity, containing somewhere between 10 to 15 million species. The tropical rain forest has four main functions; conservation of water catchments; conservation of the soil; conservation of plant and animal genetic resources; and the production of wood and other forest products. The existence of mature forests is important and vital to the survival of its habitat and to us humans. However, lately the forest scene in Malaysia and around the world has been changing quite rapidly as a result of uncontrolled or illegal logging, forest fires and pollution.

Logging is a key force driving forest degradation and biodiversity loss. Loggers in tropical Asia rarely clear cut forests. Yet they are the most important

factor triggering the process of deforestation. They selectively cut the largest and most valuable trees from primary forests. They degrade forests, leaving them more susceptible to forest fires.

Deforestation and forest fires have been especially acute in tropical Asia. From 1960 to 1980 alone, tropical Asia lost almost one third of its forest cover [16]. Deforestation continued until 1990 where the forests only covered about 350 million hectares of tropical Asia which is just half of the world's forests. [17]. By the start of the twenty-first century, Many of the countries in the Asian region were severely degraded. Currently, half of the world's humid tropical forests are already gone and now only covers 6 percent of the earths land surface.

The degradation of forests has serious implications that include accelerating biodiversity loss and climate change. These implications concern local, regional and global levels. Therefore, monitoring and managing forests in a sustainable way to reduce its degradation is crucial to the survival of tropical rainforests.

2.4 Forest Remote Sensing

Over the past few decades, remote sensing has been a valuable source of information in mapping and monitoring forest activities. As the need for increased amounts and quality of information about such activities becomes more apparent, it is felt that remote sensing as an information source will be increasingly critical in the future. The goal of a remote sensing application is the application of knowledge to solve problems [18]. There is a need to have as much relevant information as possible on the conditions of the forest to prescribe treatments, formulate policies, and to provide insight on the forest condition and health.

Remote Sensing applications include:

- Detection of deforestation activities
- Monitoring of the canopies micro climate
- Detection of changes in forest conditions

- Fire monitoring and prevention

Currently, Malaysia uses an efficient and effective forest Resources Monitoring System to monitor changes in the forest resources [19]. The system is based on a three phased approach consisting of the following:

Phase 1: The use of satellite imagery, basically Landsat TM for the establishment of a fixed grid of monitoring points over the forested area. A 2.5 minute grid has been adopted for the identification of all types of natural forest and plantation.

Phase 2: Establishment of a Geographic Information System (GIS) to describe in detail, at anytime, the past and present forest situation at these grid points. Apart from providing the information on total forested area, the system will also record changes in the forest cover, due to legal or illegal human activities.

Phase 3: Field sampling on a continuous basis, of all forest types on a randomly selected number of grid points according to predetermined accuracy standards. This sampling will keep the stand and stock table data of the various types updated.

Forest monitoring services building on data coming from earth observation satellites and in-situ sensors address the following domains:

- Climatic changes (e.g. national greenhouse gas reporting, carbon stock statistics)
- Sustainable forest management (e.g. Clear Cut mapping and monitoring service)
- Nature protection (e.g. Land Cover and Forest Indicator Service)

2.5 Sensor Node

A sensor node has one or more sensors attached that are connected to the physical world [4]. Some examples of sensors are temperature sensors, light sensors

and PIR sensors that can measure the concurrence of events in their vicinity. Thus, each sensor is a separate data source that generates records with several fields such as the identity and location of the sensor type, and the value of the reading.

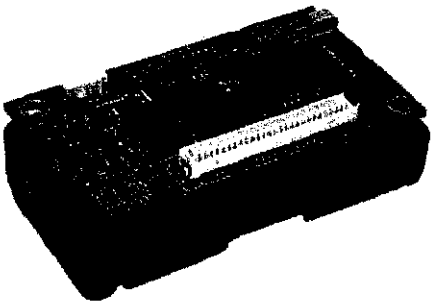


Figure 2.1: The Crossbow MICA2 Mote [20]

The architecture of a sensor node depends on the scenario in which it is being used. For instance, in a weather monitoring scenario, the sensor would need to have a temperature, humidity and light sensor. On the other hand, in an underground habitat monitoring scenario, the sensor node would probably only need heat and movement sensors to monitor the animals' activities. Figure 2.2 shows the heart of a sensor node which is the Mote board. Among the important components that are contained inside a Mote are the processor, flash memory, input and output ports and the network communication component [4].

With the Mote board being the heart of a sensor node; it still needs several other important components before it can be applied to the scenario. These components would be the all important power supply and circuitry, adapted sensors and the packaging of the node. As mentioned earlier in the report, the power management of a sensor node plays an important role in the durability of the sensor nodes in a real-world application [6]. Normally small batteries are used for a sensor node as the available size for the battery is limited by the packaging. This in turn limits the power that is available for the sensor node. Therefore, the program which runs the sensor node and the power circuitry are both optimized in order to minimize the power usage and prolong the life of the sensor node.

2.6 Summary

This chapter presents a brief overview of the technology and concepts behind Wireless Sensor Network's, the forest situation in Malaysia, current trends in forest remote sensing and the sensors used in this project. Several management plans and infrastructural layers govern the way WSNs are organized. These plans are the basis for defining the way mesh network connections within a WSN area are established and other elements in WSNs.

CHAPTER 3
METHODOLOGY

3.1 Introduction

This chapter presents the tools and applications to be used in this project. This includes TinyOS, TinyOS Simulator (TOSSIM), nesC, Java Development Kit, and MySQL.

Network Embedded System C-programming (nesC programming) was used to create the sensor node program which captures Temperature and Light (Photo) intensity readings and broadcasts the data back to the monitoring station in the form of Active Messages. The Cygwin (LINUX Emulator for Windows) environment was used for the development and compilation of the TinyOS program (written in nesC). The simulation and testing of the TinyOS program was carried out using the TinyOS Simulator (TOSSIM). The monitoring station application was developed using Java through the use of the Netbeans IDE (Integrated Development Environment). Among the features of the application are table and graph generation, real-time reporting and an interactive graphical user interface (GUI). Collected data is then stored in a MySQL database for future reuse by other systems and/or subsystems.

3.2 System Design

3.2.1 Sensor Nodes

The sensor system of RIMBAMON© consists of sensor nodes and a base station. The sensors are categorized into two types, the sensor (monitoring) nodes and

the base node. The sensor nodes will be placed throughout the forest at a specific interval from one another depending on the line of sight and the functional communication range of the sensors. Theses sensor nodes will be made up of Crossbow's MTS310CA sensor boards connected to Crossbow's MICA2 Mote's. The reason for choosing the MICA2 Mote for this application is because of its long range ISM band integrated radio which has a range of up to 500 feet [21] depending on the foliage, environmental conditions, and radio frequency. The MTS310CA sensor board offers sensing capabilities for Temperature, Light Intensity, Acoustic, Acceleration and Magnetic readings. For the implementation of this prototype, only the temperature and light intensity modules will be utilized. However, the additional sensor may also be integrated into the design of the system in the future. Therefore, this paper will also discuss the potential sensors that can be used for the monitoring of a forest area. The sensor nodes would be placed either on the trunk of the tree, near ground level or along the roadside.

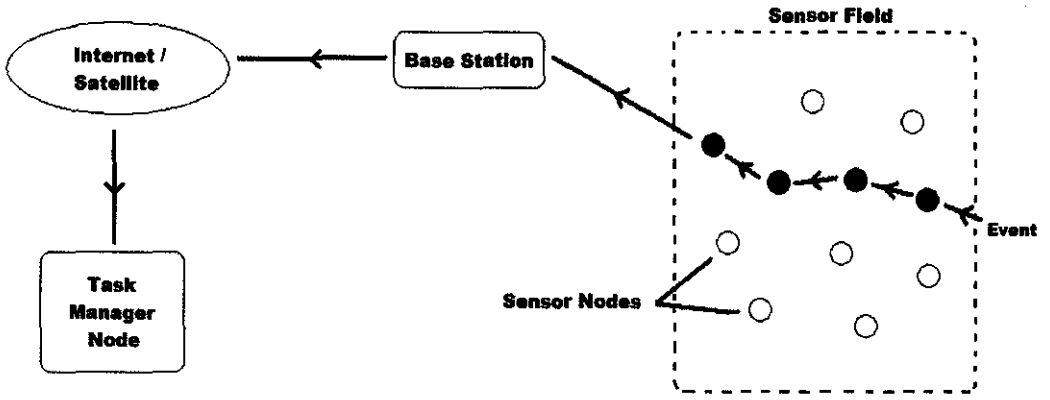


Figure 3.1: Basic Scenario for Area Monitoring

The Base node will be comprised of the same MICA2 Mote attached to the MIB510CA programmer board which offers a serial connection for data acquisition.

3.2.2 Sensor Board

The Sensor Mote may be equipped with several sensors which suits the application scenario. Among the sensors that are applicable to a system that monitors the forest conditions are the ambient temperature, the light intensity, the acoustic or

sound sensor and the air or soil humidity. However, due to the limitations of the TinyOS Simulator, only the light intensity and temperature sensors could be tested. This is due to the unavailability of other types of sensors in the simulation sensor mote model. Although the temperature and light intensity sensors may not be enough for monitoring the forest conditions, other sensors may easily be implemented in the future as the nesC programming codes for the sensor parts are similar and can be duplicated for other sensors.

1) Temperature

One of the main conditions that can directly show drastic changes in the forest would be the ambient temperature in that area. For example, any logging activity in an area would affect the surrounding temperature. The ground would heat up faster as a result of direct contact with sunlight. Consequently, hot air will rise from the ground and therefore increase the ambient temperature in that area. Although the change in temperature would be only a few degrees, it would be enough for the monitoring system to detect and display the change. Apart from logging, a sudden rise in ambient temperature would also be a warning for forest fires in that area.

2) Light

Another indication of logging and forest fires would be the ambient light intensity in that area. For the case of illegal logging, the cutting down of trees would increase the intensity of light which reaches the lower levels of the forest. Similarly, the light coming from early fires would also result in an increase of light intensity, especially during the night time. Apart from logging and forest fires, light intensity could also act as security surveillance for the forest especially during night time. If there are people in that area who use torchlight's or light a campfire, the node would report the change in light intensity and appropriate authorities could be alerted of the presence of people in that area. This not only allows monitoring of the forest during night time but also allows the efficient use of patrol forces in that area.

3) Acoustic

Acoustic or sound levels could also offer valuable information about activity in the area. In the case of illegal logging, abnormal sound levels in the area would indicate possible presence of tractors, machinery or chainsaws. Also, the sound of someone crying for help could also be detected by the sensors, thus aiding in the location of that person.

4) Humidity

Changes in the humidity level in an area would also be a tell-tale sign of heavy equipment. This is because of the heat and smoke generated by the engines. Similarly, any logging activities in nearby areas would also affect the humidity levels.

Sensor data might contain noise, and it is often possible to obtain more accurate results by fusing data from several sensors [5]. Summaries or aggregates of raw sensor data are thus more useful to sensor applications than individual sensor readings. For instance, when monitoring the concentration of a dangerous chemical in an area, one possible query is to measure the average value of all sensor readings in that region, and report whenever it is higher than some predefined threshold.

3.2.3 The nesC Program

The nesC program which will be programmed onto each sensor node consists of several parts. They are the data collector, message broadcaster, and message listener parts. The data collector is in charge of acquiring the readings from the sensor modules and parsing the information into a message for broadcasting. The sensors that will be trigger are the temperature and light intensity sensors. The second part of the program, the message broadcaster, sends a message which contains the sensor data along with other information, such as the node ID and reading count, into the network. This message will then be received by other nodes in the system and captured by the base station. The third section of the program is the message listener

which listens for messages being broadcasted inside the network and performs the appropriate response. The nesC program will be discussed in further detail later on.

3.3 Tools and Applications

Throughout this project, several tools and applications will be used. Among them are:

1. TinyOS

TinyOS is an open-source operating system designed for wireless embedded sensor networks. It features a component-based architecture which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints present in sensor networks. TinyOS's component library includes network protocols, distributed services, sensor drivers, and data acquisition tools – all of which can be used as-is or be further refined for a custom application. TinyOS's event-driven execution model enables fine-grained power management yet allows the scheduling flexibility made necessary by the unpredictable nature of wireless communication and physical world interfaces. [7]

2. TOSSIM

TOSSIM is a discrete event simulator for TinyOS sensor networks. Instead of compiling a TinyOS application for a mote, users can compile it into the TOSSIM framework, which runs on a PC. This allows users to debug, test, and analyze algorithms in a controlled and repeatable environment. As TOSSIM runs on a PC, users can examine their TinyOS code using debuggers and other development tools. [8]

3. nesC

nesC is an extension of C to support and reflect the design of TinyOS v1.0 and above. It provides a set of language constructs and restrictions to implement TinyOS components and applications. [9]

4. Java Development Kit

The Java Development Kit (JDK) is a Sun product aimed at Java developers. Since the introduction of Java, it has been by far the most widely used Java SDK. The primary components of the JDK are a selection of programming tools, including:

- **javac** – The compiler, which converts source code into Java byte code.
- **jar** – The archiver, which packages related class libraries into a single JAR file.
- **javadoc** – The documentation generator, which automatically generates documentation from source code comments.

5. MySQL

MySQL is a freely available open source Relational Database Management System [RDBMS], a database engine of sorts that uses Structured Query Language (SQL). SQL is the most popular language for adding, accessing, and processing data in a database, and is most noted for its rapid processing, proven reliability, and ease and flexibility of use. Since its release, it has quickly became the dominant web database thanks to its speed, compact size and comparative ease of use - and liberal licensing policy, being distributed from the start as open source (and later under the GPL license) and free for many purposes. [10]

3.4 Additional Tools Used

In addition to the tools and applications mentioned in the previous section, several additional tools were also used for the development of RIMBAMON©. They are the Linux emulator for Windows, Cygwin, and the Netbeans Integrated Development Environment (IDE).

3.4.1 Cygwin

Cygwin consists of a library that implements the POSIX system call API in terms of Win32 system calls, a GNU development toolchain (such as GCC and GDB)

to allow basic software development tasks, and a large number of application programs equivalent to common programs on the Unix system [11]. At this point, almost all open-source programs on Unix have been ported to Cygwin, including the X Window System, KDE, Gnome, Apache, TeX, and various others.

A mechanism has been created for installing *inetd, syslogd, sshd,* Apache and other daemons as standard Windows services, allowing a Microsoft Windows system to function much like a Unix or Linux server. All of these programs are installed through the standard Cygwin setup program, which downloads the necessary packages from the Internet. The setup program can be rerun as necessary to update programs to their latest versions or add or remove programs. Various other features are provided by setup, such as the ability to install the source code along with the binaries.

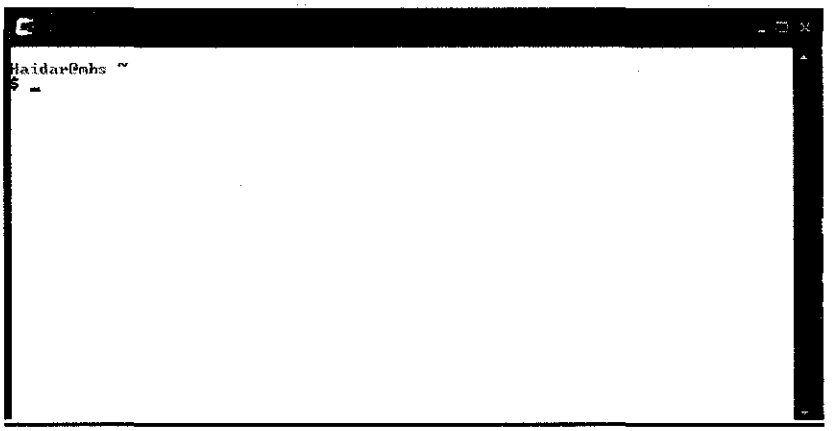


Figure 4.1: Screenshot of the Cygwin console window

3.4.2 Netbeans IDE 5.0

The NetBeans IDE is a robust, free, open source Java IDE that provides the developer with everything they need to create cross-platform desktop, web and mobile applications straight out of the box. The NetBeans IDE includes an intuitive GUI Builder (formerly known as Project Matisse) as well as comprehensive support for developing plug-in modules, and rich client applications based on the NetBeans platform.

The Netbeans IDE will be used throughout the development of the project since the program will be developed in Java. An additional feature of Netbeans that is important for the project is its GUI (Graphical User Interface) creation engine.

3.5 Summary

This chapter explained the design of the system and also the tools used in the development of the RIMBAMON© ; such as TinyOS, TinyOS Simulator (TOSSIM), nesC, Java Development Kit, MySQL, Cygwin and the Netbeans IDE.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 Introduction

This chapter presents the results and discussions that were obtained from this project, as well as the development work for the RIMBAMON© system.

4.2 Development Work

The project involves programming on TinyOS using NesC and conducting simulations using TOSSIM (TinyOS SIMulator). Throughout these past few weeks, the author has started working with TinyOS. In order to learn how the TinyOS works, the author needed to test the example source codes to familiarize himself with the TinyOS system. The following is a brief summary of the author's experience of working with TinyOS so far.

4.2.1 Installation of TinyOS

TinyOS is available for both Windows and Linux platforms. TinyOS also requires the Java SDK and the *Java Comm* package.

Installing TinyOS on the Windows platform is simple. It only requires the user to download the TinyOS 1.1.0 executable installer and execute it. The installer takes care of installing the complete TinyOS code, Java SDK and the Java Comm

package. When TinyOS is installed under Windows it automatically installs Cygwin, a Unix emulator which was mentioned earlier in the report.

Program compilation and simulation under Windows is carried out using the Cygwin libraries and console. During installation it creates a shortcut to the batch file for Cygwin on the desktop which launches the Cygwin console. Double clicking on the shortcut brings up the Cygwin terminal Window. Using this terminal Window the user is able to navigate to the TinyOS directory and compile a TinyOS application or execute TOSSIM. The files for these are located under the /opt/tinyos-1.x/ directory.

During installation the installer will automatically set and update the environment variables with TinyOS specific information under both Windows and Linux. Once installed, the user will have access to the whole TinyOS code and TOSSIM. (Information on downloading TinyOS and the detailed installation instructions is available at http://www.tinyos.net/download.html#1.1.0).

The example programs provided with TinyOS give an introduction to the TinyOS and NesC (Nested C) fundamentals. They provide a starting point to learn about some of the services provided by TinyOS and how to use these services when writing application code for sensors.

One of the many example programs that were tested is the *IntToRfm* application. It provides an introduction on how to transmit a user defined data packet of type *IntMsg* (defined in a file called *IntMsg.h*). The code for this can be found under the /opt/tinyos-1.x/tos/lib/counter/ directory (files: *IntToRfm.nc*, *IntToRfmM.nc* and *IntMsg.h*).

Similarly, the *RfmToInt* application is an example for receiving user defined packets through the transceiver. The difference is that this program reads a transmission and converts it into a data packet of type *IntMsg*. The code for this can be found in the same directory name (*RfmToInt.nc, RfmToIntM.nc, and IntMsg.h*). The code in *IntToRfm* and *RfmToInt* for sending and receiving packets is explained in lesson 4 of the TinyOS tutorials. A list of services provided by TinyOS (Components

that provide services) and interfaces used for accessing those services are provided on the TinyOS website (http://www.tinyos.net/scoop/special/support).

4.2.2 TinyOS Simulator (TOSSIM)

Lesson 5 of the tutorial introduces the user to a simulator for TinyOS called TOSSIM. To simulate the *Blink* example, first the application needs to be compiled for TOSSIM. After compilation an executable file called main.exe is created in /opt/tinyos-1.x/apps/Blink/build/pc. This exe file is the simulator which also includes the code for blink application.

Code Listing:

```
> cd /opt/tinyos-1.x/apps/Blink/ //Change directory to blink
> make pc //Compile app for TOSSIM
> export DBG=led //Instruct TOSSIM to display only led related messages
> ./build/pc/main.exe 2 //Run simulation for 2 nodes.
```

After the last line of the above code is entered in the Cygwin console, the following output is displayed in the window. These messages are repeatedly displayed until the program is stopped.

```
0: LEDS: red ON //Debug messages
1: LEDS: red OFF //Debug messages
0: LEDS: red ON //Debug messages
1: LEDS: red OFF //Debug messages
```

4.2.3 Visualization of Simulation using TinyViz

To visualize the working of a simulation, TinyOS includes a visualization tool called TinyViz. The source files for this tool are available in /opt/tinyos-1.x/tools/Java/net/tinyos/sim/. It also needs to be compiled prior to using it. To compile it, the directory is changed to /opt/tinyos-1.x/tools/Java/ and then the

command *make* is entered in the terminal window. This compiles all Java files and creates a script called TinyViz.

Code Listing:

```
> cd /opt/tinyos-1.x/tools/Java/ //Change directory to Java tools
> make //To compile all Java tools
```

To visualize the simulation of the Blink application, the following commands were used:

Code Listing:

```
> cd /opt/tinyos-1.x/apps/Blink/ //Change directory to Blink application.
> make pc //Compile Blink application.
> export DBG=led //You can specify any of the supported debug message types.
> tinyviz -run build/pc/main.exe 2 //Run TinyViz.
```

If the execution of the above code results in an error, then the PATH variable in windows has to be updated to include the TinyViz directory (/opt/tinyos-1.x/tools/java/net/tinyos/sim). This is done by adding the following line to the end of the text box labeled *Variable Value,* ";C:\tinyos\cygwin\opt\tinyos-1.x\tools\java\net\tinyos\sim". Then the Blink application is executed in a new instance of the cygwin window and TinyViz. This will display the TinyViz window with two motes.

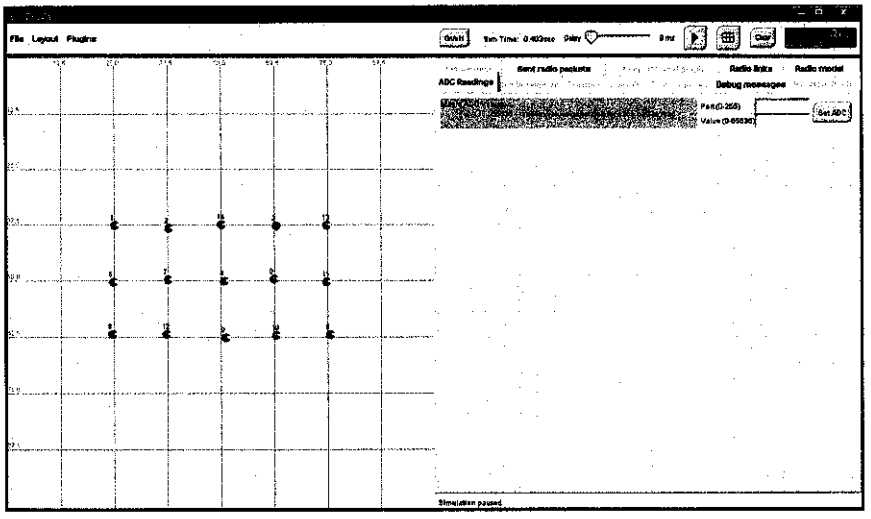


Figure 4.2: Screenshot of the TinyViz application running a simulation

4.2.4 Application Graphical User Interface (GUI)

A way to develop the system would be to use a Java GUI as the front end of the system. The GUI would enable the user to select and determine the queries and information he/she wants to see. To create a Java GUI, the author needed to research on how to develop GUI's in Java and also how to connect it to a program.

One way to create the GUI is by using Java Swing. Swing is a GUI toolkit for Java. Swing is one part of the Java Foundation Classes (JFC). It includes graphical user interface (GUI) widgets such as text boxes, buttons, split-panes, and tables.

Swing widgets provide more sophisticated GUI components than the earlier Abstract Window Toolkit. Since they are written in pure Java, they run the same on all platforms, unlike the AWT which is tied to the underlying platform's windowing system. Swing supports pluggable look and feel – not by using the native platform's facilities, but by roughly emulating them. This means the user can get any supported look and feel on any platform. The disadvantage of lightweight components is possibly slower execution. The advantage is uniform behavior on all platforms.

For the simplicity of building a GUI, the Netbeans IDE's built-in GUI builder will be used for this project. Through the use of an established GUI builder, less time is required on learning the fundamentals required to program a Swing GUI. The development of the GUI would be started only when the basic components of the project such as the data retrieval and logging are completed.

4.2.5 MySQL

The MySQL database has become the world's most popular open source database because of its consistent fast performance, high reliability and ease of use [10]. MySQL is used in more than 10 million installations ranging from large corporations to specialized embedded applications on every continent in the world.

There are APIs available that allow applications written in numerous programming languages to access MySQL databases, including: C, C++, C#, Borland Delphi (via dbExpress), Eiffel, Smalltalk, Java (with a native Java driver implementation), Lisp, Perl, PHP, Python, Ruby, REALbasic (Mac), FreeBasic, and Tcl; each of these uses a specific API. An Open Database Connectivity (ODBC) interface called MyODBC allows additional programming languages that support the ODBC interface to communicate with a MySQL database, such as ASP or Coldfusion. MySQL is mostly implemented in ANSI C.

Working with databases and SQL programming is also quite new to the author. At the time of the report is written, the author is in the process of learning and developing the data logging and storing component of the RIMBAMON© .

In addition to the testing of the TinyOS example applications, the author also performed a literature review on the topics related to the project. These included research papers regarding other findings on the development of query processing in Sensor Networks. Also the author used the various manuals and tutorials on TinyOS, TinyDB, TOSSIM and Cygwin.

4.3 Results

A prototype of the system has been successfully developed. This prototype consists of 2 parts, namely the TinyOS program and the Java-based Monitoring Station Application. The TinyOS program is programmed onto the sensor motes and functions by broadcasting the captured readings in the form of Active Messages and enters sleep mode until the next event is triggered. This is to prolong the battery life in order to minimize maintenance. The Monitoring Station Application provides a graphical interface that displays critical information in the form of tables, graphs and location maps. Messages received from the sensor motes are in hexadecimal values. Therefore, the monitoring system converts this message into meaningful values. This application utilizes a database storage system to store the captured data for future retrieval.

4.3.1 Conversion of Raw Sensor Data

The values received from the raw message are hexadecimal values arranged in *little endian* format where bytes are switched around. This means that if the bytes from the raw message shows 06 01, then the actual value is 01 06. This process is done inside the OscopeMsg.class.

For the calculation of the Temperature in engineering units (Celsius), the following formula is implemented [20]:

$$1/T(K) = a + b \times \ln(R_{thr}) + c \times [\ln(R_{thr})]^3$$

$$T(\text{in Celsius}) = T(\text{in Kelvin}) - 273$$

where:

$R_{thr} = R1(ADC\_FS-ADC)/ADC$

$a = 0.00130705$

$b = 0.000214381$

$c = 0.000000093$

$R1 = 10$ k.

$ADC\_FS = 1023$

ADC = output value from Mote's ADC measurement.

The range of the temperature sensor is from 0^o to 50^o Celsius. The light intensity is calculated using the following formula;

$$Light = Battery\_Voltage \times (ADC / 1023)$$

4.3.2 nesC Program

The author has also worked on the nesC programming of the sensor motes. The program will need to collect samples of temperature and the light percentage and

then pack the data in a message and transmit it back to the base station. The application will need to be able to receive and detect AM messages coming from within the network. For messages that are received by the base mode, the readings will be forwarded to the base station. Each sensor mote will receive the same message but only the messages received at the base node would be retrieved and processed. The testing of the program was done using the nesC compiler and the TOSSIM simulator to verify the operation of the code.

The program is based on the Oscilloscope.nc, OscilloscopeM.nc and OscopeMsg.h files of the OscilloscopeRF application that come with the TinyOS installation. The oscilloscope program is a demo to show the sensing capabilities of the sensor motes running the TinyOS. The first step for modifying the program would be to create another data stream for the temperature readings in the transmitted message format. This is because the existing data stream only holds data for the light sensors. This was done by adding the following line to the OscopeMsg.h file;

```
uint16_t dataPhoto[BUFFER_SIZE];
uint16_t dataTemp[BUFFER_SIZE];
```

Then the temperature sensing operation would need to be added to the oscilloscope main code file. This is done by declaring the Temperature sensor for the Analog-to-Digital unit on the sensor;

```
OscilloscopeM.SensorControl -> Photo;
OscilloscopeM.SensorControl -> Temp;
OscilloscopeM.Photo -> Photo.PhotoADC;
OscilloscopeM.Temp -> Temp.TempADC;
```

The capturing of the sensor readings is done inside the OscilloscopeM.nc file. The method for this is to first call the Light sensor operation, followed by the Temperature sensing. In order to do this, the temperature sensing method needs to be called after the light sensing method has finished. The code for achieving this is presented below;

```
 async event result_t Photo.dataReady(uint16_t data) {
  struct OscopeMsg *pack;
  atomic {

   pack = (struct OscopeMsg *)msg[currentMsg].data;
   pack->dataPhoto[packetReadingNumber++] = data;

   readingNumber++;
   dbg(DBG_USR1, "data_event\n");
       call Temp.getData();
   }
  return SUCCESS;
 }

// added code for temperature sensing
 async event result_t Temp.dataReady(uint16_t data2) {
  struct OscopeMsg *pack;
  atomic {
   pack = (struct OscopeMsg *)msg[currentMsg].data;
   pack->dataTemp[packetReadingNumber2++] = data2;
   readingNumber++;
   dbg(DBG_USR1, "data_event\n");
   if (packetReadingNumber2 == BUFFER_SIZE) {
       }
  }
  return SUCCESS;
 }
```

As seen in the above, the call Temp.getData(); is called from within the light sensing operation. Therefore, the temperature sensing operation is done after the light sensing operation. The reason for this is because the motes share the same ADC pin [20]. Therefore, the readings need to be taken one after another and not simultaneously. The readings are packed into the transmitted message by the following lines;

```
pack->dataPhoto[packetReadingNumber++] = data;
pack->dataTemp[packetReadingNumber2++] = data2;
```

In order to take 5 samples, each time before sending the message out, a loop is implemented to repeat the sampling process until the message buffer is full. The packetReadingNumber keeps track of the number of samples which has already been added to the message buffer. After the message has been sent, the packetReadingNumber is reset to 0 for the next sampling round.

The triggering of the sampling process is done by sending a command message from the base node to the appropriate sensor nodes. This way, the battery life of the sensor node could be preserved even more as the nodes do not perform anything until the command message is received. The sending of the command message is done by using the MIG (Message Interface Generator). This is done by first implementing the MoteIF interface. This interface represents a Java interface for sending and receiving messages to and from motes.

```
// OK, connect to the serial forwarder and start receiving data
mote = new MoteIF(PrintStreamMessenger.err, oscilloscope.group_id);
```

Then, the message is sent by invoking the MoteIF.send() together with the destination address and message. Here, MoteIF.TOS_BCAST_ADDR is used to represent the broadcast destination address, which is identical to TOS_BCAST_ADDR used in the nesC code. This destination address can also be replaced with the target sensor node's ID in the network.

```
try {
  mote.send(MoteIF.TOS_BCAST_ADDR, new OscopeResetMsg());
} catch (IOException ioe) {
  System.err.println("Warning: Got IOException sending reset message: "+ioe);
  ioe.printStackTrace();
}
```

The section of the nesC program which responds to the issued command message is presented below. A variable trigger is used to indicate when a command message is received and is accessible to other portions of the program which depends on this variable. Here, the value of trigger is set to 1 when the message is received. A change in the status of the Red LED is initiated each time a command message is received.

```
event TOS_MsgPtr ResetCounterMsg.receive(TOS_MsgPtr m) {
        call Leds.redToggle();
        trigger=1;
  return m;
 }
```

After the command message is received and the variable trigger is set to 1, the sampling of the sensors begins. The sampling procedure is encapsulated inside a loop which is only executed when the previously explained trigger value is at 1. Therefore, the mote does virtually nothing as long as the command message is not received. The sampling of the sensor data is repeated until the packetReadingNumber reaches the 5 samples buffer limit of the message. As soon as this value is reached, the sending of the message is triggered. The function that is responsible for the broadcasting of the message is called dataTask() and will be explained later on. After the message is done with the broadcasting, the loop is exited and the mote waits for the next command message to arrive. The following is the excerpt of the explained code;

```
 event result_t Timer.fired() {
        dbg(DBG_USR1, "timer\n");
        if(trigger==1){
                call Photo.getData();
                if (packetReadingNumber == BUFFER_SIZE) {
                post dataTask();
                call Leds.greenToggle();
          }
    }
  return SUCCESS;
```

```
}
```

Broadcasting of the message is done by the dataTask() function. This function inserts the necessary data into the message format and broadcasts it into the network. This is done by calling the pack syntax. For example, pack->sourceMoteID = TOS_LOCAL_ADDRESS; inserts the Node's ID in to the sourceMoteID field in the OscopeMSG format. Apart from packing data into the message, this function also resets the variable packetReadingNumber to 0. As explained earlier, this variable defines the number of samples taken before triggering the dataTask() function. All the necessary information is packed into the message before sending it out.

```
if (call DataMsg.send(TOS_BCAST_ADDR,sizeof(struct OscopeMsg), &msg[currentMsg]))
 {
     atomic {
      currentMsg ^= 0x1;
     }
     call Leds.yellowToggle();
 }
     trigger=0;
```

The above code shows the implementation of the DataMsg.send() function which performs the message broadcasting. In order for the message to be successfully broadcasted, several values need to be inserted together with the function call. They are the TOS_BCAST_ADDR value which is the address where the message is to be broadcast to, sizeof(struct OscopeMsg) which is the size of the message being sent and lastly the message itself. Once the message is successfully broadcasted, the message count is increased and the Yellow LED is toggled to indicate the operation is completed. Finally, the variable trigger which was explained earlier is set to 0 to restart the command message waiting process.

After the program code writing was completed, the next step was testing of the code. Testing is by first compiling the source in Cygwin, using the make pc command (the target make pc, is designated for builds targeted for simulation with

TOSSIM), and then simulating the code using TOSSIM. The following is an example of the messages that were detected by the Data Capturing System;

```
Message <OscopeMsg>
 [sourceMoteID=0x2]
 [lastSampleNumber=0x1e]
 [channel=0x1]
 [dataPhoto=0x139 0x102 0x160 0x106 0x83 ]
 [dataTemp=0x115 0x116 0x2cf 0x393 0x87 ]

Light(%): 31 (0x139) ; Temperature(Celsius): 2 (0x115) ;
Light(%): 25 (0x102) ; Temperature(Celsius): 2 (0x116) ;
Light(%): 34 (0x160) ; Temperature(Celsius): 48 (0x2cf) ;
Light(%): 26 (0x106) ; Temperature(Celsius): 87 (0x393) ;
Light(%): 13 (0x83) ; Temperature(Celsius): 0 (0x87) ;
```

As displayed above, the broadcasted message contains similar header fields as the original Oscilloscope program but with addition parameters. They are the Photo and Light sensor readings. The application takes 5 readings for each sensor before transmitting the message. However, the resulting values in this example seem to be illogical. This is because of the random ADC values generated by the simulator, TOSSIM. For more accurate simulation of the scenario, the ADC values need to be manually added to the TinyViz interface.

4.3.3 GUI Interface

An important feature of the monitoring station application would be its Graphical User Interface (GUI). The GUI would enable the user to select and view the results and information he/she wants to see. To create a Java GUI, the author needed to do research on how to develop GUI's in Java and also how to integrate coding with the interface.

One way was by using Java Swing. Swing is a GUI toolkit for Java and is part of the Java Foundation Classes (JFC). It includes graphical user interface (GUI) widgets such as text boxes, buttons, split-panes, and tables.

Swing widgets provide more sophisticated GUI components than the earlier Abstract Window Toolkit . Since they are written in pure Java, they run the same on all platforms, unlike the AWT which is tied to the underlying platform's windowing system. Swing supports pluggable look and feel – not by using the native platform's facilities, but by roughly emulating them. This means the user can get any supported look and feel on any platform. The disadvantage of lightweight components is possibly slower execution. The advantage is uniform behavior on all platforms.

The Netbeans IDE was used in the development of the java program and interface. It eases the process of constructing the GUI. The built-in GUI builder enables the user to graphically design and edit a user interface. This important feature not only avoids the complex workings of building a GUI, but also reduces the time spent on building the interface. Therefore, additional time would be available for the development and tweaking of the rest of the system.

Main Window

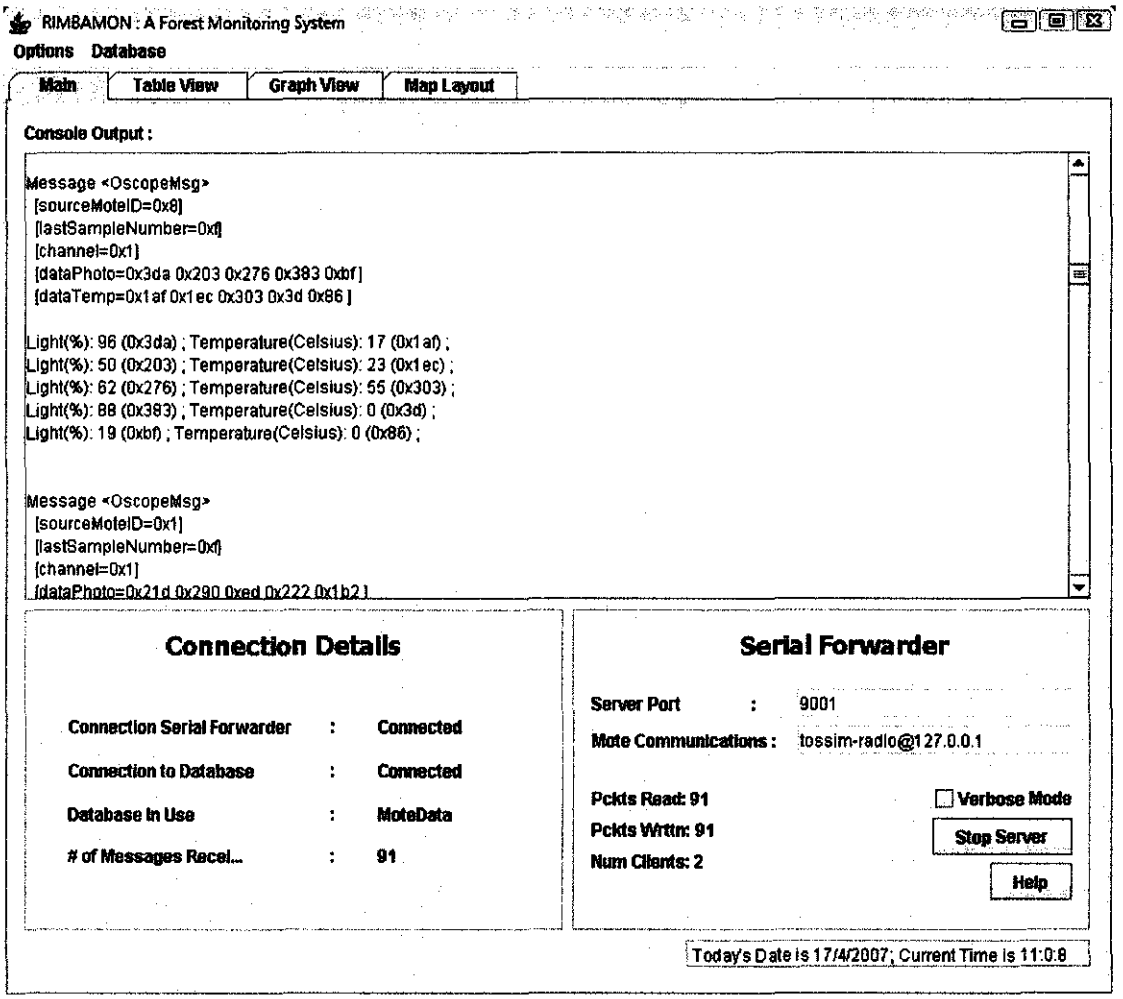
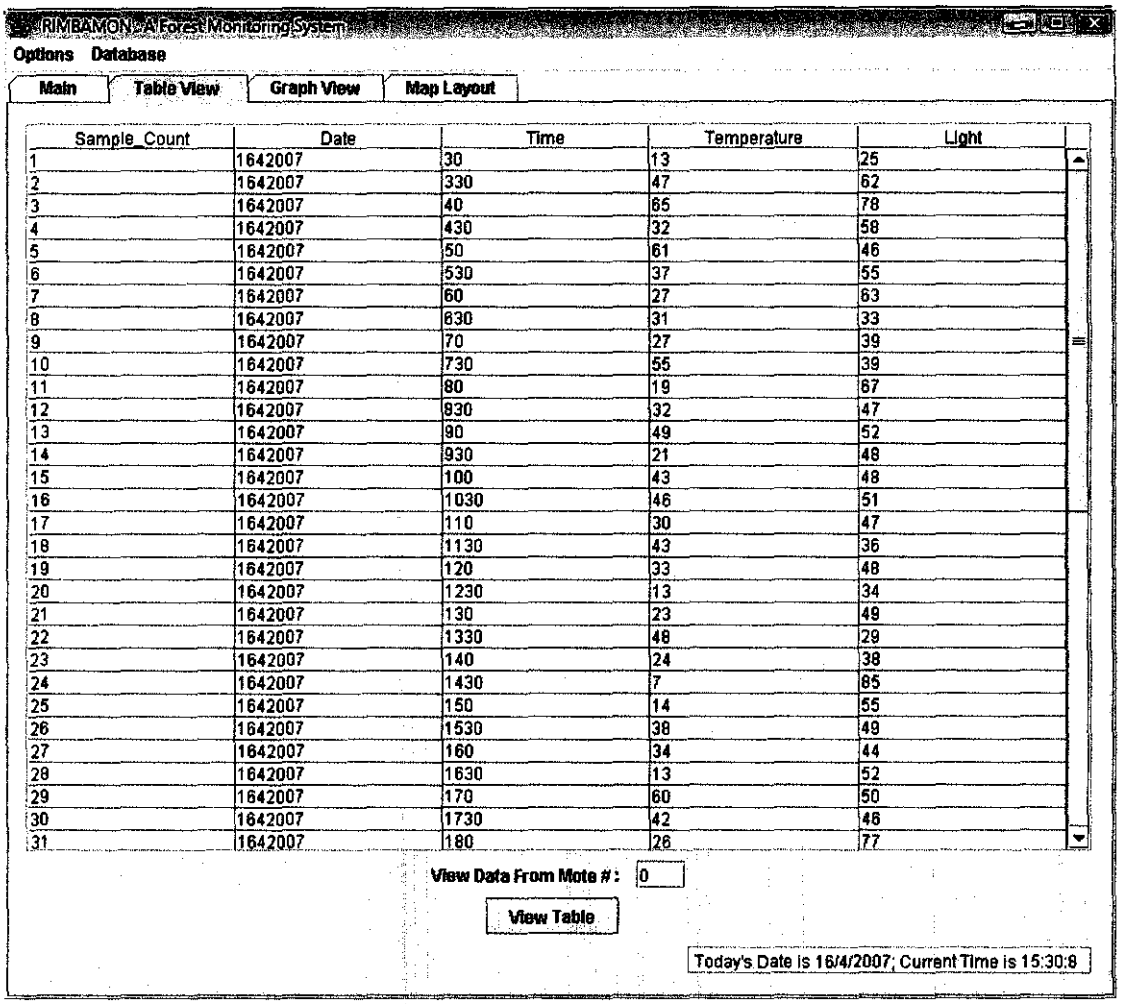


Figure 4.3: The Main Window – Connection Details and Console output area

This is the first window that shows when the program is started. The main window provides connection information and provides a console view of the information being passed through. The two connections that are displayed are the database connection and the SerialForwarder connection. In the database connection area, the status of the connection and the database which is in use is displayed. In the Serial Forwarder connection area, the server's port and the mode of connection are displayed. The number of packets received from the motes is shown in the *Pckts Read* field. Each time a message is received, the value in the field will be increased by 1. Similarly, the number of messages sent by the program to the motes are also recorded and shown in the *Pckts Wrttn* field. The console view area displays the program's output which includes errors, raw messages received, and status messages.

Table View Window



Sample_Count	Date	Time	Temperature	Light
1	1642007	30	13	25
2	1642007	330	47	62
3	1642007	40	65	78
4	1642007	430	32	58
5	1642007	50	61	46
6	1642007	530	37	55
7	1642007	60	27	63
8	1642007	630	31	33
9	1642007	70	27	39
10	1642007	730	55	39
11	1642007	80	19	67
12	1642007	830	32	47
13	1642007	90	49	52
14	1642007	930	21	48
15	1642007	100	43	48
16	1642007	1030	46	51
17	1642007	110	30	47
18	1642007	1130	43	36
19	1642007	120	33	48
20	1642007	1230	13	34
21	1642007	130	23	49
22	1642007	1330	48	29
23	1642007	140	24	38
24	1642007	1430	7	65
25	1642007	150	14	55
26	1642007	1530	38	49
27	1642007	160	34	44
28	1642007	1630	13	52
29	1642007	170	60	50
30	1642007	1730	42	46
31	1642007	180	26	77

Figure 4.4: Table View Window– Displays the data from a selected Mote

This window displays the information or sensor data that has been gathered from the motes. The data is grouped into individual database tables according to the ID of the nodes. A MySQL database is used to store the data. The user specifies the data of which node is to be displayed by entering the node ID into the text field at the bottom. By pressing the View Table button after the node ID has been entered, a table showing the data for that node will be displayed. The table is made up of 3 sections which are the Sample Count, Temperature Data and Light Intensity Data. The table provides the user with instant data retreival and enables the user to browse through the data to view specific samples. The Temperature readings are represented in Celsius whereas the Light Intensity readings are in Percentage (%).

Graph View Window

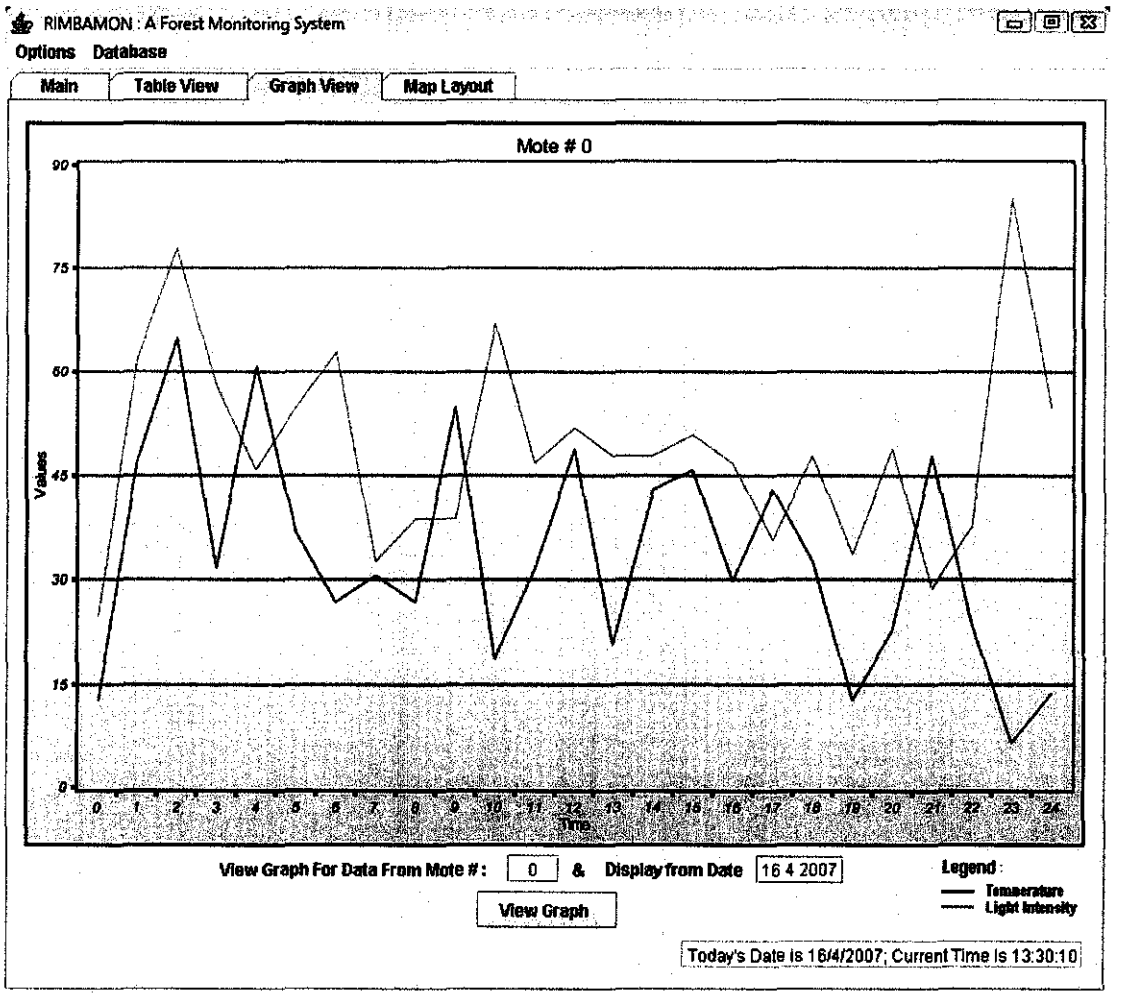


Figure 4.5: Graph View Window – Line Graph representation of sensor data.

The Graph View Window displays the selected data in a line graph format. The user selects the node and the frame limit of the samples and presses the View Graph Button to display the graph. The graph shows the node ID at the top followed by the nodes data. This data is retrieved from the same database used for the Table View Window. Data is represented in two lines, one for temperature and another for light intensity. The Temperature readings (Values) are represented in Celsius whereas the Light Intensity readings are in Percentage (%).

Map Layout Window

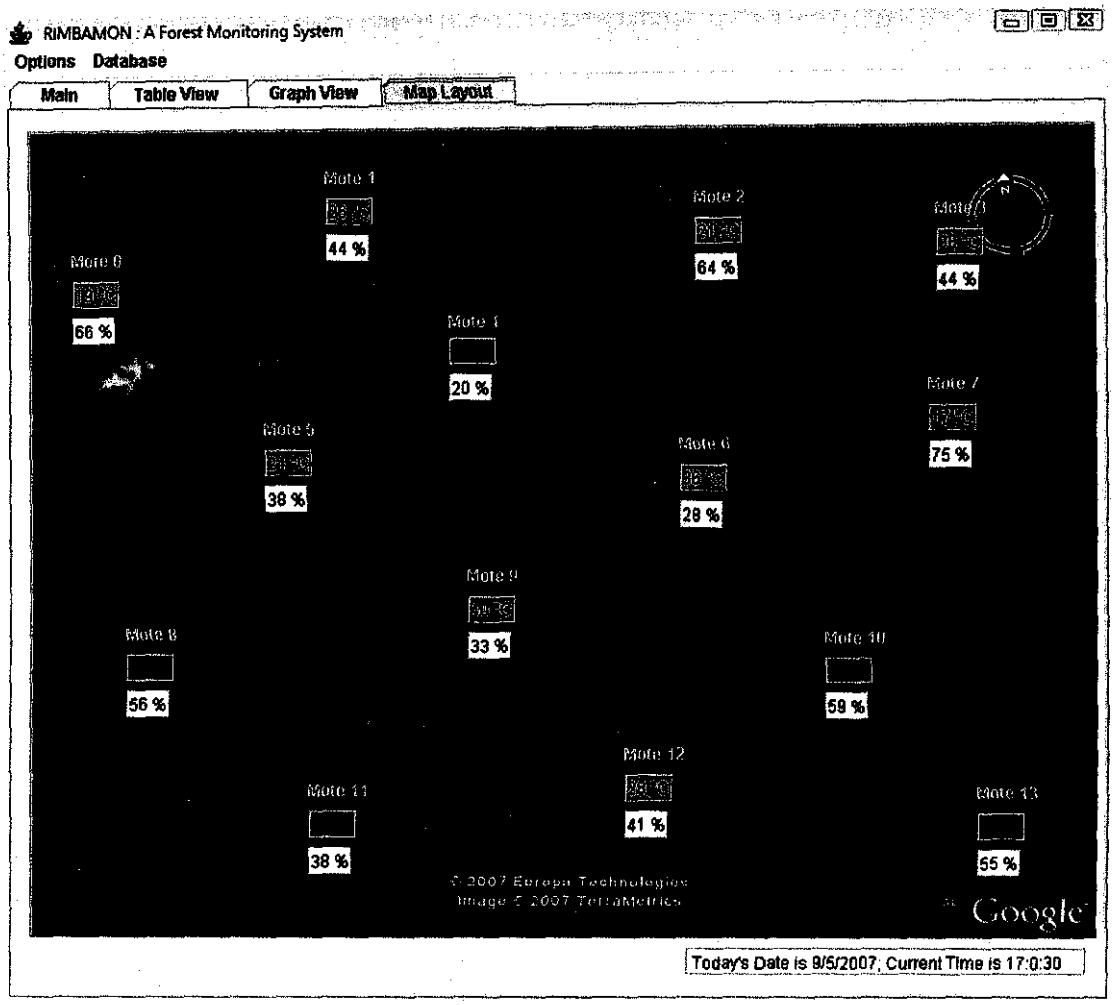


Figure 4.6: Map Layout – Displays the location of the motes in the area.

This window shows a graphical representation of the location of the nodes throughout the forest area, along with real-time display of the sensor data. The data displayed is updated each time a new set of sensor readings is received by the program. Whenever the program detects a reading which is beyond a certain threshold, a warning sign (represented by a red color) is shown at the concerned mote's ID and location. This will help alert the operator which is monitoring the program. This would also aid in the dispatching of forest patrol teams to the area as the exact location had already been identified.

4.3.4 Monitoring Station Application

The program consists of several parts which work together to perform the tasks described and shown earlier. The following table briefly describes the functions of each java class and its relevance to other classes;

Class Name	Function(s)
Main.class	○ Main class ○ Starts Capture.class
Capture.class	○ Create connection to SerialForwarder ○ Logging of messages and activity ○ Create connection to MySQL database ○ Handle messages coming from SerialForwarder ○ Convert Raw data into proper SI representations
DBaccess.class	○ Generate table in the Table View Window ○ Retrieve specific data from database based on user input
Graphaccess.class	○ Generate graph in the Graph View Window ○ Retrieve specific data from database based on user input
OscopeMsg.class	○ Contains format of incoming raw messages ○ Parses data from the raw message ○ Enables retrieval of specific data in raw message ○ Generated by the Message Interface Generator (MIG)
OscopeResetMsg.class	○ Defines format for the Reset command message ○ Generated by the Message Interface Generator (MIG)
SFClient.class	○ Listens for requests from a connected Aggregator Server ○ Send messages retrieved from motes to the serial port
SFListen.class	○ Spawns SerialPortReader and ServerReceivingThread
SerialForwarder.class	○ Handles initiation of SerialForwarder ○ Handle interface elements
TinyUI.class	○ Main Graphical User Interface (GUI) class ○ Defines design of GUI ○ Creates externally accessible GUI components (Buttons, Labels)

Table 4.1: Description of class files

The main functions of the program are to:

1) Listen to data coming from the serial port (COMX) or TOSSIM

- The data would be taken by listening to the TOSSIM outputs and its reports. However, in a real life application, the system would be listening through the serial port which is connected to the base station for the sensor motes. The X indicates the communications port number of the computer which is connected to the base station.

2) Capture the streaming raw data

- The raw data that is captured is in a complex and hard-to-understand format, hexadecimals. The raw message consists of a stream of hexadecimal bytes arranged in the *little endian* format. The message format will be explained later on. This part of the program only functions to capture the raw data and provide it to the application when requested.

3) Manipulate the data

- This part of the system would be in charge of converting the raw data into an understandable or human-friendly format for later reporting and also storing of the data in a database. To accomplish this, certain algorithms or functions are applied to the raw data. These algorithms or formulae are among the information provided together with the data sheet of the mote. At this stage, the data is ready for representation in graphs or tables and also for storing.

4) Store the data

- The converted data would then be stored in a database for later reference and recalling. For this part of the system, a MySQL database would be used. The database file would be stored on the host computer and copied to create backups in case of data loss. Through the use of an MySQL database, the important information can be protected with passwords to prevent unauthorized access.

5) Report results

- The converted and stored data will be recalled for reporting. The specific data to be recalled will be determined by the user. He/she would be able to select the data

to be displayed by selecting the options in the system's GUI. The Map Layout Window provides the user with real-time sensor data from the sensors. The data is represented in the form of graphs and tables.

4.4 Summary

This chapter described the development work, and discusses the results or outcome of the development work. Among the work that has been done are learning how to use Cygwin and Netbeans, Installation and setting up of TinyOS, TOSSIM, TinyViz, TinyDB and MySQL. The result of the development work is a forest monitoring system that consists of two separate programs, namely the sensor node program and the monitoring station program. This system, called RIMBAMON©, provides a method of monitoring the changes in the microenvironment under the forest canopy.

CHAPTER 5
CONCLUSION AND RECOMMENDATIONS

Simulation and testing of RIMBAMON© have shown that it meets the expectations and aims previously set at the beginning of the project. RIMBAMON© has proved its capabilities to capture sensor data, broadcast the data using radio communication, capture sensor readings coming from the sensor motes, convert the Hexadecimal-based messages into meaningful data, represent the data in graphs, tables and maps and store the data in a database. RIMBAMON© offers an alternative to existing but costly aerial and satellite surveillance systems. The representation of data in the form of line graphs, tables and location maps may help with surveillance to prevent illegal logging activities and provide early warning for forest fires. Integration of additional sensors and surveillance hardware to the existing basic system of RIMBAMON© is possible and can certainly be further developed into an integrated Forest Decision Support System (DSS).

Future possibilities for the RIMBAMON© system include increasing the monitoring area by increasing the number of sensor motes inside the system. Additionally, new sensor capabilities such as seismic, sound and humidity sensing devices can be added to the TinyOS program that has been developed in the basic system. Remote or web monitoring can also be implemented by integrating a GSM module into the basic system of RIMBAMON© and monitoring can be carried out remotely using the internet. The possible implementations of a system such as RIMBAMON© is not limited only to forest monitoring. The system can be modified for application in oil rigs and processing plants. However, proper encapsulation of the sensors would need to be built in order to protect the sensors from extreme conditions. This project provides an insight into the capabilities of Wireless Sensor Technology in the area of environmental, habitat and industrial monitoring.

It is hoped that findings of this research may provide a better understanding of the issues and possibilities of Wireless Sensor Networks data and information management.

REFERENCES

[1] Kay Romer, Friedamann Mattern (2004), ***The Design Space of Wireless Sensor Networks***. . Last access: 30th May 2007. Citing Internet sources URL http://www.vs.inf.ethz.ch/publ/papers/wsn-designspace.pdf

[2] Wireless Sensor Network article. Last access: 27th April 2007. Citing Internet sources URL http://en.wikipedia.org/wiki/Sensor_Networks

[3] Baptiste Pretre (2005), ***Sensor Networks : Exposure Problem***. Last access: 30th May 2007. Citing Internet sources URL http://www.vs.inf.ethz.ch/edu/SS2005/DS/reports/09.1-sensornetze-report.pdf

[4] Zhan Yi (2005). ***An Introduction of Wireless Sensor Networks (WSN) Technology***. Last access: 30th May 2007. Citing Internet sources URL http://www.sasase.ics.keio.ac.jp/jugyo/2005/report-WSN.pdf

[5] D. L. Hall, J. Llinas (2001), ***Handbook of Multisensor Data Fusion***. CRC Press.

[6] David Culler, Deborah Estrin, Mani Srivastava (2004, August). ***Overview of Sensor Networks***. *Computer*, 41-49. Last access: 30th May 2007. Citing Internet sources URL http://www.archrock.com/downloads/resources/IEEE-overview-2004.pdf

[7] ***TinyOS: Mission Statement***. Last access: 30th May 2007. Citing Internet sources URL http://www.tinyos.net/special/mission

[8] Philip Levis, Nelson Lee (2003). ***TOSSIM : A Simulator for TinyOS Networks.*** Last access: 30th May 2007. Citing Internet sources URL http://www.cs.berkeley.edu/~pal/pubs/nido.pdf

[9] Eric Brewer, David Culler, David Gay, Phil Levis, Rob von Behren, Matt Welsh. Last access: 27th April 2007. Citing Internet sources URL http://nescc.sourceforge.net/

[10] MySQL, Last access: 27th April 2007. Citing Internet sources URL http://sql-info.de/mysql/mysql.html

[11] Cygwin article. Last access: 27th April 2007. Citing Internet sources URL http://en.wikipedia.org/wiki/Cygwin

[12] TinyOS Tutorials. Last access: 27th April 2007. Citing Internet sources URL http://www.tinyos.net/tinyos-1.x/doc/tutorial/index.html

[13] Sam Madden, Joe Hellerstein, Wei Hong (2003), ***TinyDB : In-Network Query Processing in TinyOS***. Last access: 30th May 2007. Citing Internet sources URL http://telegraph.cs.berkeley.edu/tinydb/tinydb.pdf

[14] Feng Zhao, Leonidas Guibas (2004). ***Wireless Sensor Networks: An Information Processing Approach***. Morgan Kaufmann Publishers.

[15] Anna Hac (2003). ***Wireless Sensor Network Designs***. New York : John Wiley & Son.

[16] Peter Dauvergne, ***Loggers and degradation in the Asia Pacific***, Cambridge University Press.

[17] Sharma, Narend P.,ed., 1992. ***Managing the Worlds Forests : Looking for Balance Between Conservation and Development***, Washington DC: World Bank

[18] Steven E. Franklin (2001) ***Remote Sensing for Sustainable Forest Management***, Lewis Publishers

[19] ***Tropical Forest Mapping and Monitoring in Malaysia***, 1999, Global Observation of Forest Cover Workshop (GOFC). Last access: 30th May 2007. Citing Internet sources URL http://www.eoc.ukm.my/forest.pdf

[20] ***MPR-MIB Users Manual, Revision B***, June 2006. Crossbow Technology Inc. Last access: 30th May 2007. Citing Internet sources URL http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf

[21] ***Environmental Monitoring Application Note***, Crossbow Technology Inc. Last access: 30th May 2007. Citing Internet sources URL http://www.xbow.com/Support/Support_pdf_files/Smart_Dust_AppNote.pdf?sid=76

APPENDICES

APPENDIX A
MAIN.JAVA PROGRAM CODE

***The following is an excerpt of the java program code for the main.java file. This file is part of the RIMBAMON© system and is the intellectual property of the author. Therefore, please contact the author for enquiries regarding the full program code.**

```
.
.
.
.
public static void main(String[] args) throws IOException {
    if (args.length == 1) {
        group_id = (byte) Integer.parseInt(args[0]);
        System.err.println("oscilloscope: Using group ID "+group_id);
        System.err.println("Note: group id should not be specified if you're using a TOSBase base station");
       }
    new SerialForwarder(args);
    app = new Main();
app.init();
  app.start();

}
.
.
.
.
```

APPENDIX B
CAPTURE.JAVA PROGRAM CODE

***The following is an excerpt of the java program code for the Capture.java file. This file is part of the RIMBAMON© system and is the intellectual property of the author. Therefore, please contact the author for enquiries regarding the full program code.**

```
.
.
.
.
//Register the JDBC driver for MySQL.
Class.forName("com.mysql.jdbc.Driver");

//Define URL of database server for
// database named mysql on the localhost
// with the default port number 3306.
String url = "jdbc:mysql://localhost/mysql";

//Get a connection to the database for a
// user named root with a blank password.
// This user is the default administrator
// having full privileges to do anything.
boolean conStatus = false;

do{
        try{
          final Connection con = DriverManager.getConnection(url,"root", "");
          stmt = con.createStatement();
          conStatus = true;
          stmt.executeUpdate("USE MoteData;");
          TinyUI.labelDBName.setText("MoteData");
          TinyUI.msgArea.append("\n**********************************\n");
          TinyUI.msgArea.append("  Connect to database server !!  ");
          TinyUI.msgArea.append("\n**********************************\n");
          TinyUI.labelConDB.setText("Connected");

          TinyUI.bRefreshTable.addActionListener(new java.awt.event.ActionListener(){
            public void actionPerformed(ActionEvent e){
              DBaccess frame=new DBaccess(con,stmt);
            }
          });
.
.
.
.
//Received raw values and converting
        for (i = 0; i < limit; i++) {
          int val_Photo = omsg.getElement_dataPhoto(i);
        int val_Temp = omsg.getElement_dataTemp(i);

        double val_Temp2 = 1*val_Temp; // to convert int to double
        double Rthr = 10000 * ( (1023 - val_Temp2) / val_Temp2 );
        double a = 0.001010024, b = 0.000242127, c = 0.000000146;
        double ln_Rthr = java.lang.Math.log(Rthr);
        double pow_ln_Rthr = (java.lang.Math.pow(ln_Rthr,3));
        double temp_Kelvin = a + (b * ln_Rthr) + (c * pow_ln_Rthr);
        double Kelvin = 1/temp_Kelvin;
```

```
        double temp_Celsius = Kelvin - 273;
        long Celsius = java.lang.Math.round(temp_Celsius) ;
        if (Celsius < 0 ) Celsius = 0 ;

        double val_Photo2 = 1*val_Photo; // to convert int to double
        double temp_Photo = (val_Photo2/1023) * 100;
        long Photo = java.lang.Math.round(temp_Photo) ;

        dbcount++;
        tempFoto+=Photo;
        tempTemp+=Celsius;
        if(dbcount==5){
        Celsius2 = tempTemp/5;
        Photo2 = tempFoto/5;
        stmt.executeUpdate("INSERT INTO mote" + moteID + " (Date,Time,Temperature,Light) VALUES " +
             "('" + currentDate +  "','" + currentTime + "','" +Celsius2 +
             "','"+ Photo2 +"')");
        dbcount=0;

//Temperature reaction
        if (moteID==0) {
          TinyUI.fieldMote0Temp.setText(Celsius2+" \u00b0C");
          if(Celsius2>40) {
            TinyUI.fieldMote0Temp.setBackground(new java.awt.Color(255, 51, 51));
            TinyUI.msgArea.append("\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
            TinyUI.msgArea.append("\n!! Warning Temp Level detected at Mote0 !!");
            TinyUI.msgArea.append("\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
          } else TinyUI.fieldMote0Temp.setBackground(new java.awt.Color(204, 204, 255));
        }
.
.
.
.
      //Light Reaction
               if (moteID==0) {
        TinyUI.fieldMote0Light.setText(Photo2+" %");
        if(Photo2>50 && (clock.h<7 || clock.h>19)) {
          TinyUI.fieldMote0Light.setBackground(new java.awt.Color(255, 51, 51));
          TinyUI.msgArea.append("\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
          TinyUI.msgArea.append("\n!! Warning Light Level detected at Mote0 !!");
          TinyUI.msgArea.append("\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
        }
        else TinyUI.fieldMote0Light.setBackground(new java.awt.Color(255, 255, 204));
      }
.
.
.
.
```

APPENDIX C
DBACCESS.JAVA PROGRAM CODE

***The following is an excerpt of the java program code for the DBAccess.java file. This file is part of the RIMBAMON© system and is the intellectual property of the author. Therefore, please contact the author for enquiries regarding the full program code.**

```
.
.
.
.
public class DBaccess {

  public DBaccess(Connection con, Statement stmt){
    try{
      TinyUI.msgArea.append(">> Accessed Database!  \n");
      stmt.executeUpdate("USE MoteData;");
      String userEntry = TinyUI.fieldChooseTable.getText();
      String query= ("select * from mote"+userEntry+"");
      rs=stmt.executeQuery(query);
      model=new ScrollingResultSetTableModel(rs);
      TinyUI.dataTable.setModel(model);
    }catch(SQLException e){
      System.out.println("Error "+e);
    }
  }
.
.
.
.
```

APPENDIX D
GRAPHACCESS.JAVA PROGRAM CODE

***The following is an excerpt of the java program code for the GraphAccess.java file. This file is part of the RIMBAMON© system and is the intellectual property of the author. Therefore, please contact the author for enquiries regarding the full program code.**

```
.
.
.
.
public Graphaccess(int moteNum1, String toNum1) {
    try{
        Statement stmt;

        //Register the JDBC driver for MySQL.
        Class.forName("com.mysql.jdbc.Driver");

        //Define URL of database server for
        // database named mysql on the localhost
        // with the default port number 3306.
        String url = "jdbc:mysql://localhost/mysql";

        //Get a connection to the database for a
        // user named root with a blank password.
        // This user is the default administrator
        // having full privileges to do anything.
        Connection con = DriverManager.getConnection(url,"root", "");

        //Get a Statement object
        stmt = con.createStatement();
        stmt.executeUpdate("USE MoteData;");
        int moteNum = moteNum1;
        String sampleFromString = "5";
        int from = Integer.parseInt(sampleFromString);
        int temp1 = Integer.parseInt(toNum1);
        System.out.println(temp1);
        ResultSet srs = stmt.executeQuery("select Temperature,Light from mote"+moteNum+" where
          Date="+temp1);
        srs.last();                          // Jump to last row
        int rowcount = srs.getRow();         // get the row count
        srs.beforeFirst();                   // reset to allow forward cursor processing
        System.out.println(rowcount);
.
.
.
.
    }
.
.
.
.
}
```

APPENDIX E
SFCLIENT.JAVA PROGRAM CODE

***The following is an excerpt of the java program code for the SFClient.java file. This file is part of the RIMBAMON© system and is the intellectual property of the author and the authors mentioned in the file. Therefore, please contact the author for enquiries regarding the full program code.**

```
// $Id: SFClient.java,v 1.6.2.2 2003/08/18 22:09:43 cssharp Exp $

/*                                                                    tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 */


/**
 * File: ServerReceivingThread.java
 *
 * Description:
 * The ServerReceivingThread listens for requests
 * from a connected Aggregator Server.  If a data
 * packet is received, it is sent on to the serial
 * port.
 *
 * @author <a href="mailto:bwhull@sourceforge.net">Bret Hull</a>
 * @author <a href="mailto:dgay@intel-research.net">David Gay</a>
 *
 */
.
.
.
.
```

```
public class SFClient extends SFProtocol implements Runnable, PacketListenerIF {
   private Thread thread;
   private Socket socket = null;
   private SerialForwarder sf;
   private SFListen listenServer;
.
.
.
.
   private void readPackets() throws IOException {
            for (;;) {
               byte[] packet = readPacket();

               sf.incrementPacketsWritten();
               if (!listenServer.source.writePacket(packet))
                        sf.verbose.message("write failed");
      }
   }

   public void packetReceived(byte[] packet) {
            try {
               writePacket(packet);
            }
            catch (IOException e) {
//             shutdown();
            }
   }
}
```

APPENDIX F
SFLISTEN.JAVA PROGRAM CODE

***The following is an excerpt of the java program code for the SFListen.java file. This file is part of the RIMBAMON© system and is the intellectual property of the author and the authors mentioned in the file. Therefore, please contact the author for enquiries regarding the full program code.**

```
// $Id: SFListen.java,v 1.6.2.3 2003/08/26 17:40:25 idgay Exp $
/*                                                                tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 */
/**
 * File: ListenServer.java
 *
 * Description:
 * The Listen Server is the heart of the serial forwarder.  Upon
 * instantiation, this class spawns the SerialPortReader and the
 * Multicast threads.  As clients connect, this class spawns
 * ServerReceivingThreads as wells as registers the new connection
 * SerialPortReader.  This class also provides the central
 * point of contact for the GUI, allowing the server to easily
 * be shut down
 *
 * @author <a href="mailto:bwhull@sourceforge.net">Bret Hull</a>
 * @author <a href="mailto:dgay@intel-research.net">David Gay</a>
 */

.
.
.
.

public class SFListen extends Thread implements PacketListenerIF, PhoenixError {
  PhoenixSource source;
```

```
  private ServerSocket serverSocket;
  private Vector clients  = new Vector();
  private SerialForwarder sf;
.
.
.
.
  public void run() {
        try {
          sf.message("Listening to " + sf.motecom);

          source = BuildSource.makePhoenix(sf.motecom, sf.verbose);
          if (source == null) {
                sf.message("Invalid source " + sf.motecom + ", pick one of:");
                sf.message(BuildSource.sourceHelp());
                return;
          }
          source.setPacketErrorHandler(this);
          source.registerPacketListener(this);
          source.start();

          // open up our server socket
          try {
                serverSocket = new ServerSocket(sf.serverPort);
        TinyUI.labelConSF.setText("Connected");
          }
          catch (Exception e) {
                sf.message("Could not listen on port: " + sf.serverPort);
                source.shutdown();
        TinyUI.labelConSF.setText("Error");
                return;
          }

          sf.verbose.message("Listening for client connections on port " + sf.serverPort);
          try {
                for (;;) {
                  Socket currentSocket = serverSocket.accept();
                  SFClient newServicer =
                        new SFClient(currentSocket, sf, this);
                  clients.add(newServicer);
                  newServicer.start();
                }
          }
          catch (IOException e) { }
        }
    finally {
//          cleanup();
      sf.verbose.message("----------------------");
    }
  }
.
.
.
.
  public void packetReceived(byte[] packet) {
        sf.incrementPacketsRead();
  }
}
```

APPENDIX G
SERIALFORWARDER.JAVA PROGRAM CODE

***The following is an excerpt of the java program code for the SerialForwarder.java file. This file is part of the RIMBAMON© system and is the intellectual property of the author and the authors mentioned in the file. Therefore, please contact the author for enquiries regarding the full program code.**

```
// $Id: SerialForwarder.java,v 1.2.2.4 2003/09/12 15:00:41 mdwelsh Exp $

/*                                                                  tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 */

/**
 * File: SerialForwarder.java
 *
 * Description:
 * The SerialForwarder class provides many static functions
 * that handle the initialization of the serialforwarder
 * and/or the associated gui.
 *
 * @author <a href="mailto:bwhull@sourceforge.net">Bret Hull</a>
 * @author <a href="mailto:dgay@intel-research.net">David Gay</a>
 */
.
.
.
.
public class SerialForwarder implements Messenger {
  // appication defaults

  public SFListen listenServer;
```

```
  public static String motecom = "tossim-radio@127.0.0.1";
  public boolean logDB;

  public static int serverPort = 9001;

.
.
.
.
  public void message(String msg) {
        if ( TinyUI.msgArea != null) {
          TinyUI.message(msg);
        }
        else {
          System.err.println(msg);
        }
  }

  synchronized public void incrementPacketsRead() {
    nPacketsRead++;
    if (TinyUI.msgArea != null) {
          TinyUI.updatePacketsRead(nPacketsRead);
        }
  }

  synchronized public void incrementPacketsWritten() {
    nPacketsWritten++;
    if (TinyUI.msgArea != null) {
          TinyUI.updatePacketsWritten(nPacketsWritten);
        }
  }

  synchronized public void incrementClients() {
    nClients++;
    if (TinyUI.msgArea != null) {
          TinyUI.updateNumClients(nClients-1);
        }
  }

  synchronized public void decrementClients() {
    nClients--;
    if (TinyUI.msgArea != null) {
          TinyUI.updateNumClients(nClients);
        }
  }

  public synchronized void startListenServer() {
        if (listenServer == null) {
          nClients = nPacketsRead = nPacketsWritten = 0;
          listenServer = new SFListen(this);
          listenServer.start();
        }
        if (TinyUI.msgArea != null) {
          TinyUI.updateListenServerStatus(true);
          TinyUI.updateNumClients(nClients);
          TinyUI.updatePacketsWritten(nPacketsWritten);
          TinyUI.updatePacketsRead(nPacketsRead);
        }
  }
}
```

APPENDIX H
TINYUI.JAVA PROGRAM CODE

***The following is an excerpt of the java program code for the TinyUI.java file. This file is part of the RIMBAMON© system and is the intellectual property of the author. Therefore, please contact the author for enquiries regarding the full program code.**

```
.
.
.
.
public class TinyUI extends javax.swing.JFrame {

    private SerialForwarder sf;

    /** Creates new form TinyUI */
    public TinyUI() {

        initComponents();
    }
.
.
.
.
// Variables declaration
    public javax.swing.JButton bHelp;
    public static javax.swing.JButton bRefreshGraph;
    public static javax.swing.JButton bRefreshTable;
    public static javax.swing.JButton bStopServer;
    public static javax.swing.JCheckBox cbVerboseMode;
    public static javax.swing.JTable dataTable;
    public static javax.swing.JTextField fieldChooseGraph;
    public static javax.swing.JTextField fieldChooseTable;
    public static javax.swing.JTextField fieldMoteCom;
    public static javax.swing.JTextField fieldServerPort;
    public static javax.swing.JTextField fieldGraphTo;
    public static javax.swing.JTextField fieldMote0Light;
    public static javax.swing.JTextField fieldMote0Temp;
    public static javax.swing.JTextField fieldMote10Light;
    public static javax.swing.JTextField fieldMote10Temp;
    public static javax.swing.JTextField fieldMote11Light;
    public static javax.swing.JTextField fieldMote11Temp;
    public static javax.swing.JTextField fieldMote12Light;
    public static javax.swing.JTextField fieldMote12Temp;
    public static javax.swing.JTextField fieldMote13Light;
    public static javax.swing.JTextField fieldMote13Temp;
    public static javax.swing.JTextField fieldMote1Light;
    public static javax.swing.JTextField fieldMote1Temp;
    public static javax.swing.JTextField fieldMote2Light;
    public static javax.swing.JTextField fieldMote2Temp;
    public static javax.swing.JTextField fieldMote3Light;
    public static javax.swing.JTextField fieldMote3Temp;
    public static javax.swing.JTextField fieldMote4Light;
    public static javax.swing.JTextField fieldMote4Temp;
    public static javax.swing.JTextField fieldMote5Light;
    public static javax.swing.JTextField fieldMote5Temp;
    public static javax.swing.JTextField fieldMote6Light;
    public static javax.swing.JTextField fieldMote6Temp;
```

```
public static javax.swing.JTextField fieldMote7Light;
public static javax.swing.JTextField fieldMote7Temp;
public static javax.swing.JTextField fieldMote8Light;
public static javax.swing.JTextField fieldMote8Temp;
public static javax.swing.JTextField fieldMote9Light;
public static javax.swing.JTextField fieldMote9Temp;
public static javax.swing.JTextField jTextField2;
public static javax.swing.JTextField fieldClock;
public static javax.swing.JTextField fieldClock1;
public static javax.swing.JTextField fieldClock2;
public static javax.swing.JTextField fieldClock3;
public javax.swing.JLabel jLabel1;
public javax.swing.JLabel jLabel10;
public javax.swing.JLabel jLabel11;
public javax.swing.JLabel jLabel12;
public javax.swing.JLabel jLabel13;
public javax.swing.JLabel jLabel14;
public javax.swing.JLabel jLabel15;
public javax.swing.JLabel jLabel16;
public javax.swing.JLabel jLabel17;
public javax.swing.JLabel jLabel18;
public javax.swing.JLabel jLabel19;
public javax.swing.JLabel jLabel2;
public javax.swing.JLabel jLabel6;
public javax.swing.JLabel jLabel7;
public javax.swing.JLabel jLabel8;
public javax.swing.JLabel jLabel9;
public javax.swing.JMenu jMenu1;
public javax.swing.JMenu jMenu2;
public javax.swing.JMenu jMenu3;
public javax.swing.JMenuBar jMenuBar1;
public javax.swing.JMenuBar jMenuBar2;
public javax.swing.JMenuItem jMenuItem1;
public javax.swing.JMenuItem jMenuItem2;
public javax.swing.JPanel jPanel1;
public javax.swing.JPanel jPanel2;
public javax.swing.JPanel jPanel3;
public javax.swing.JPanel jPanel4;
public javax.swing.JPanel jPanel5;
public static javax.swing.JPanel jPanel6;
public javax.swing.JPanel jPanel7;
public javax.swing.JPanel jPanel8;
public javax.swing.JScrollPane jScrollPane1;
public javax.swing.JScrollPane jScrollPane2;
public javax.swing.JTabbedPane jTabbedPane1;
public static javax.swing.JLabel labelConDB;
public static javax.swing.JLabel labelConSF;
public static javax.swing.JLabel labelDBName;
public static javax.swing.JLabel labelMoteCom;
public static javax.swing.JLabel labelMsgRcv;
public static javax.swing.JLabel labelNumClients;
public static javax.swing.JLabel labelPacketsReceived;
public static javax.swing.JLabel labelPacketsSent;
public static javax.swing.JLabel labelServerPort;
public static javax.swing.JTextArea msgArea;
// End of variables declaration
.
.
.
}
```