M-LOC: REAL TIME LOCATION SYSTEM FOR WIRELESS MESH NETWORKS

By

Abdiel Avilés-Jiménez

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER ENGINEERING

University Of Puerto Rico
Mayagüez Campus
2008

Approved by:

_____	_____
Manuel Rodríguez-Martínez, PhD Member, Graduate Committee	Date
_____	_____
Pedro Rivera-Vega, PhD Member, Graduate Committee	Date
_____	_____
Bienvenido Vélez-Rivera, PhD President, Graduate Committee	Date
_____	_____
Dorothy Bollman, PhD Representative of Graduate Studies	Date
_____	_____
Isidoro Couvertier-Reyes, PhD Chairperson of the Department	Date

ABSTRACT

M-LOC: REAL TIME LOCATION SYSTEM FOR WIRELESS MESH NETWORKS

By

Abdiel Avilés-Jiménez

This thesis presents the design and implementation of a real time location system (RTLS) for wireless mesh networks. The system uses the received signal strength as the method of distance measurement. A prototype of this system is presented using off-the-shelf components. Arguments are made for its advantages over current alternatives in terms of development and deployment costs. Several experiments were conducted on an experimental prototype and a simulated environment. These tests evidence the capabilities and functionality of the system to locate nodes on a wireless network.

RESUMEN

M-LOC: SISTEMA DE LOCALIZACION EN TIEMPO REAL PARA REDES INALAMBRICAS EN TOPOLOGIA DE MALLA

Por

Abdiel Avilés-Jiménez

Este trabajo de tesis presenta el diseño e implementación de un sistema de localización en tiempo real para redes inalámbricas en topología de malla. El sistema utiliza el indicador de fuerza de señal de radio frecuencia como el método para determinar distancias. Se presenta también un prototipo del sistema utilizando tecnología comercial como parte de las herramientas para diseño y desarrollo. Se argumenta sobre los beneficios de estas decisiones sobre la facilidad de desarrollo y costos de instalación. También se han conducido varias pruebas sobre este prototipo experimental. Estas pruebas demostraron las capacidades y funcionalidad del sistema.

To God and my family …

ACKNOWLEDGEMENTS

Thank you God for guiding me throughout the years of my life, and for giving me the strength and courage to finish this work.

I would like to thank my beautiful family, my dad Rafael, my mom Aurea and my brothers Kevin and Hugo. I would have never started, continued or finished without your love and support. Thank you for believing in me and the decisions I have made. A special thanks to my advisor Professor Bienvenido Velez for being the first one to give me an opportunity as a research student and to continue supporting me through the years of my bachelors and masters degrees. Thank you for giving me the confidence and motivation I needed to pursue graduate studies.

Finally I want to thank in the most profound way all my good friends at ADMG's lab. Thank you Angel, Oliver, Lucho, Juddy, Ricardo, Jose Javier, Pablo and many others, for your friendship, knowledge, support, coffee, and countless midnights hours of companionship.

To all of you, I dedicate this work.

Table of Contents

ABSTRACT ..... **II**

RESUMEN ..... **III**

ACKNOWLEDGEMENTS ..... **VI**

TABLE OF CONTENTS ..... **VII**

FIGURE LIST ..... **IX**

1 INTRODUCTION **10**

1.1 MOTIVATION 11
1.1.1 Applications *11*
1.1.2 Market Expectations *13*
1.2 GPS COMPARISON 13
1.3 PROPOSED SOLUTION 16
1.4 OBJECTIVES OF THIS RESEARCH 16
1.5 CONTRIBUTIONS 17
1.6 STRUCTURE OF THIS DISSERTATION 18

2 RELATED WORK **19**

2.1 MEASUREMENT TECHNIQUES 19
2.1.1 Received Signal Strength *19*
2.1.2 Time of Arrival *20*
2.1.3 Angle of Arrival *20*
2.1.4 Connectivity *20*
2.2 LOCATION ESTIMATION ALGORITHMS 21
2.2.1 Triangulation *21*
2.2.2 Hyperbolic Trilateration *24*
2.2.3 Cooperative Location Topologies *24*
2.2.4 Ranging *26*
2.3 RELATED TECHNOLOGIES: MOVING OBJECTS DATABASES 27

3 RTL INFRASTRUCTURE **28**

3.1 DESIGN REQUIREMENTS 28
3.1.1 Digi XBee Wireless Module *30*
3.1.2 TI MSP430F1232 MCU *31*
3.1.3 Hardware Interface *32*
3.2 ZIGBEE MESH NETWORKING PROTOCOL 34
3.3 RSS INDICATOR 35
3.4 CENTRALIZED LOCATION 37

4 RTL ALGORITHMS **41**

4.1 TRILATERATION ALGORITHM 41
4.2 RSS TO DISTANCE RELATION CHARACTERIZATION 44
4.3 WIRELESS NODES FIRMWARE 47
4.3.1 API Mode *47*
4.3.2 Wireless Node Main Algorithm *48*

4.3.3 RSS Sensing *52*
4.3.4 Broadcast Frame ID List *54*
4.4 CENTRAL SERVER APPLICATION 55
4.4.1 XBee Software Interface *56*
4.4.2 Cooperative Trilateration *57*
4.4.3 User Interface *63*
4.5 SIMULATION ENVIRONMENT 66

5 EXPERIMENTAL ANALYSIS 68

5.1 INTRODUCTION 68
5.2 PROTOTYPE IMPLEMENTATION ANALYSIS 69
5.2.1 Prototype Location *71*
5.3 EFFECTS OF NETWORK SIZE IN THE LOCATION ERROR 72
5.3.1 Distributed Location Error Propagation *72*
5.4 LOCATION LATENCY 76
5.5 COST AND POWER ANALYSIS 77

6 CONCLUSIONS 80

6.1 SUMMARY OF CONTRIBUTIONS 80
6.2 FUTURE WORK 83

Figure List

Table 1.1 GPS versus M-Loc Comparison 15
Figure 2.1 Common circle patterns. (a) Type 1 (b) Type 2 (c) Type 3 22
Figure 2.2 Uncommon circle patterns 23
Figure 3.1 Network Block Diagram 29
Figure 3.2 Modules Block Diagram 30
Figure 3.3 Hardware Block Diagram 33
Figure 3.4 PC – Coordinator Interface 33
Figure 3.5 Self Healing Mesh Network 35
Figure 3.6 Location Propagation 38
Figure 3.7 Special Case 39
Figure 4.1 Trilateration 42
Figure 4.2 Distance vs. RSS characterization 46
Figure 4.3 General XBee API Frame Structure 48
Figure 4.4 Broadcast Frame 49
Figure 4.5 Wireless Node main algorithm pseudo-code 50
Figure 4.6 RSS sensing pseudo-code 52
Figure 4.7 PWM RSS signal 53
Figure 4.8 Frame ID search and store 54
Figure 4.9 Simplified XBee Software Interface Class Diagram 56
Figure 4.10 Broadcast and Return Data Flow 58
Figure 4.11 Central Server Distributed Trilateration Simplified Algorithm 60
Figure 4.12 Central Server Simplified Class Diagram 62
Figure 4.13 M-Loc Prototype 64
Figure 4.14 Wireless Nodes. (a) Beacon 1, (b) Coordinator, (c) Beacon 2, d) Unknown 65
Figure 4.15 Simulation Environment 67
Figure 5.1 Experimental antennas setting 70
Figure 5.2 Position error at different broadcasts measurements 71
Figure 5.3 Simulation Environment 74
Figure 5.4 Average position error by network size 75
Figure 5.5 Location Latency 76

1 INTRODUCTION

Node location in wireless mesh networks is an indispensable requirement for a whole array of applications. Node location has applications in network routing optimization, asset tracking, asset identification and other areas. The availability of low-cost and low power sensor networks has made an accurate, low-power and low cost scheme for node location possible. This kind of peer-to-peer, or mesh, network provides an inherent flexibility in terms of network topology. The location technique and algorithm should be modeled based on the requirements of the application such as accuracy, energy efficiency, scalability, or cost. There are various distance measurement techniques that could be used for wireless mesh network location, the most practical ones for our particular interest being time-of-arrival (TOA), angle-of-arrival (AOA), and received-signal-strength (RSS) [1]. Based on low-cost, low-power and availability constrains, RSS will be the target of our research.

Most current solutions to the location problem are addressed by using a GPS (Global Positioning System) device. This approach is suited to some applications, yet it increases the device cost, device size, and limits its applications. Our solution has several additional advantages over a GPS device. A GPS module by itself does only one thing, determine its location. Our solution solves the same problem in a different scenario involving low-cost, low-power, mesh-networked, indoors/outdoors wireless nodes. Also, our system takes advantage of the inherent collaboration among devices to lower the costs of communication between end devices and the server by not using costly high powered central antennas.

A low-cost device is a key characteristic of the success of our approach, since our target is cattle monitoring and theft prevention and similar applications. These kinds of applications need practical devices suited to the task. The devices must be as small as possible, easy to hide, easy to maintain, easy to replace, easy to install, and readily available.

1.1 Motivation

Following the computation era comes the ubiquitous computing era, an age where we might not be able to discern from futile and intelligent objects. An age where devices work and make decisions for us without us even knowing or commanding it. This is the place where wireless sensor networks are guiding us to. Wireless sensor networks are clusters of nodes that interact with each other to share gathered information, execute commands and most importantly to communicate. Within the vast ocean of applications of wireless sensor networks, our research focuses on sensor networks that determine their physical location. There are several approaches to solve this problem of sensor location

1.1.1 Applications

Here we present a minimal subset of applications and possibilities for wireless mesh network node location.

1.1.1.1 Animal Tracking and Theft Prevention

This application can be very useful in biological research, cattle monitoring and pet location. Animal behavior and interaction can be easily characterized by continuously acquiring location data for each specimen. Cattle monitoring is another important area in the

industry for theft prevention as well as health and data collection. No different are the needs of pet owners which could greatly benefit by knowing in real time their pet location at all times.

1.1.1.2 Personnel Tracking

As stated before, people location can be critical for the operations of businesses and events. A hospital is a good example where knowing who and where the closest doctor or nurse is can decide between life and death. An almost endless list of applications related to this area could be enumerated.

1.1.1.3 Logistics

Business organizations can benefit from asset location to improve their supply chain logistics and processes. Office and warehouse equipment could be monitored on temperature, humidity, location and other important information to the processes. Another example where logistics can be improved is on massive events like concerts, theatrical pieces, movie productions, and any other event where personnel location could significantly help achieve the goals of the event.

1.1.1.4 Industrial / Home Automation

Home appliances can adjust, react and collaborate by being aware of people's locations as well as other appliances and home equipment.

1.1.1.5 Robotics

In many applications robots need to be aware of the location of other robots for numerous reasons. For instance, in space exploration, a swarm of robots could be deployed in a vast area to efficiently collect information. All of their interactions with their environment

and themselves depend on their awareness of location to maximize their collaboration and use of scarce resources.

1.1.2 Market Expectations

IDTechEx firm develops various reports on RFID (Radio Frequency Identification) and related technologies. Their latest report [3] on Real Time Location Systems analyses the technologies, the market and related issues. IDTechEx has constructed a ten year forecast in which they state that the active RFID market will grow to over 11 times its present size by 2017. Active RFID technologies relate to Real Time Locating Systems (RTLS), Ubiquitous Sensor Networks, active RFID tags based on ZigBee, and Ultra Wide Band and WiFi. They predict that the active RFID market "will rise from 12.7% of the total RFID market this year to 26.3% in 2017", which translates to a $7.07 billion market.

1.2 GPS Comparison

Location estimation can be addressed by various technologies as previously mentioned. However, not all solutions to a problem are viable alternatives to specific applications. In his survey paper J. Hightower[15] developed a study in which current alternatives to the problem are compared side by side. We can observe by his analysis that currently there is no single solution to the location estimation problem. We must then look at our problem as a composition of problems and requirements and then apply a solution that fits them all.

We want to address a specific application, cattle monitoring and theft prevention. This problem involves, variable scalability, limited budget, and limited maintenance among other limitations. Our mayor concerns are location accuracy and cost. Most current solutions to the location problem are addressed by using a GPS (Global Positioning System) device. This approach is suited to some applications, yet it increases the device cost, device size, and these disadvantages limit its applications.

Our solution offers several advantages over GPS devices, including those previously mentioned. A GPS module by itself only does one thing, determine its location. After locating itself, that information must be transmitted to a centralized station where that information it is useful. This is accomplished by adding a link to the central station, usually a wireless transceiver. Our approach tries to solve the problem by using the network itself to achieve object location in order to minimize costs. This does reduces costs but also may decreases the location accuracy. One of our objectives is to quantify this potential decrease in accuracy and verify its adequacy for our particular application.

It is clear that the system must be based over a communications infrastructure. This communication infrastructure could be based on nodes composed of a wireless transceiver and a microcontroller. This platform is viable for both our approach as well as a GPS solution.

Solution	Accuracy In Meters	Device Cost	Physical Limitations	Trade Off
GPS	5 m[1]	$94 to $130[1]	not indoors external antenna	3.5 times increase on cost
M-Loc	12 m[2]	$25 to $40[2]	decreased accuracy	2.4 times worst location accuracy

[1] Data obtained from http://www.deluoelectronics.com and based on their GPS modules.
[2] Data obtained from experimental results on chapter 5 of this document. 12% of 100m radius.

Table 1.1 GPS versus M-Loc Comparison

Table 1.1 shows a simplified comparison of both approaches, GPS versus M-Loc. Comparing both approaches we observe that a GPS solution improves the location accuracy by 2.4 times of that obtained by M-Loc, but it increases the costs by 3.5 times. There is a trade off between system costs and location accuracy. A final judgment must be based on the system requirements and constrains. Since our target application is cattle theft prevention, we believe that our approach better fits our application.

It is important to clarify the accuracy and scalability of these systems. It is known that a GPS device can operate anywhere around the world, thus this might be interpreted as a worldwide coverage. This is not correct since the coverage of a real time location system is determined by the wireless transceiver that shares the information with the central station. The information is useless at the nodes. So the accuracy must be determined based on the coverage area, or area of operation. Having similar communication infrastructures for both approaches, their coverage area is of about 100m radius per node. M-Loc accuracy is dependent of its coverage area, not being the case of a GPS-based system.

1.3 Proposed Solution

We want to build a basic, low cost RTLS based on wireless mesh networks. The specifics of the network can be modeled based on each application, and each application is influenced by the market. The hardware infrastructure must be built upon off-the-shelf components. These decisions must be based on future expectations and current adoption of the hardware products. So our focus is on generic software, and innovative applications and uses for the existing infrastructure.

In a nutshell, our solution creates a software middleware for real-time location based on wireless, low-cost, low-power, mesh-networked devices.

1.4 Objectives of this Research

The main objectives of this research are:

- To design a real time location system based on a wireless mesh network using the received signal strength as the method of distance measurement.
- To develop a prototype real time location system using low cost off-the-shelf components
- To develop an extensible middleware for real time location systems and related applications.

- To develop a simulator and conduct experiments to measure the performance of our proposed solution for RTLS as the number of nodes increases.
- To investigate the accuracy of RTLS based on wireless mesh networks and compare it with alternative approaches.

1.5 Contributions

The major contributions of this work can be summarized as follows:

- Built a prototype to real time location system based on wireless mesh network using off-the-shelf technology.
- Implemented a software middleware for RTLS.
- Implemented a simulations framework to test RTLS methodologies and algorithms.
- Conducted experiments with our prototype demonstrating that the system locates nodes in the network within a 12% average accuracy of its node signal coverage radius.
- Conducted experiments on a simulation environment to validate the correctness of our algorithms demonstrating that the system is capable of locating nodes within an 18.5% average accuracy regardless of the network size.

- Conducted experiments with our prototype and measured a location latency of 1.69 seconds on average.

1.6 Structure of this Dissertation

The remainder of this dissertation is structured as follows. In Chapter two we present a survey of the most relevant previous research related to our work, namely the different techniques for measuring the distance and direction of a wireless signal and various location estimation algorithms for wireless nodes. In chapter three we present an overview of the proposed solution, a real time location system for wireless mesh networks. Chapter four discusses the implementation details of the proposed solution and in chapter five we present the results from several experiments measuring the performance of the system. Finally chapter six presents a summary of contributions, conclusions and directions for future work.

2 RELATED WORK

2.1 Measurement Techniques

The location of a node in a wireless system involves the measurement and collection of information from radio frequency signals traveling between the target nodes and a minimum of location aware reference nodes, called beacons [4]. Depending on the technique and available information we could use the received signal strength (RSS), time of arrival (TOA), difference on time of arrival (TDOA), angle of arrival (AOA), or a combination of some of these to estimate the location of nodes.

2.1.1 Received Signal Strength

Received signal strength (RSS) is defined as the sensed or measured power metric of the last received transmission by a node. The loss on the signal can be used to approximate the distance between the measuring nodes. It is widely used as a measurement technique since the hardware that implements it consists of very simple and inexpensive circuitry [5] and it is available on most commercial transceivers. Yet these types of measures suffer from unpredictability in the received measurement. This is due to many sources of error. These can be mitigated using several techniques that will be described in section 3.3, making RSS a feasible alternative.

2.1.2 Time of Arrival

Another measuring technique is the time of arrival (TOA) and difference in time of arrival (TDOA). TOA is simply the time of transmission of a message plus a propagation time delay. This measured delay between sender and receiver is used to estimate their separation distance divided by the propagation velocity. The signal could be acoustic or RF. The propagation speed for RF is considered to be ten times as fast as the speed of sound. Sensors need to have clocks that are accurately synchronized to determine the time delay by subtracting the known transmit time from the measured. A common practice is to measure the round trip time, since an asynchronous sensor can be used. A constant delay from the re-sender might be included in the equation.

2.1.3 Angle of Arrival

The angle of arrival (AOA) of the signals can be used as complementary location information to TOA and RSS measurements. It works by using an array of sensors capable of determining the direction of an incoming distance. This method can be made extremely accurate if sufficient antennas or sensors are used. The obvious drawback to this technique is the increase in device cost. Although by itself this technique cannot determine distance, it can be used to dramatically improve TOA or RSS calculations.

2.1.4 Connectivity

Connectivity is a simple binary value that only tells whether another device is within its range or not [6]. This kind of information could be used for simple routing protocols. It

might be able to aid in location if the nodes are capable of controlling their signal range and makes a relation between transmitted power and distance.

2.2 Location Estimation Algorithms

Although this review is about cooperative location algorithms, we must first present the basics of wireless node location. We start by reviewing the basics of triangulation and trilateration. This is the starting point of all the following algorithms and techniques.

2.2.1 Triangulation

Triangulation is a method that estimates the coordinates $(\bar{x}, \bar{y})$ of a point by solving a set of linear equations involving the coordinates of multiple reference points (Xi, Yi) at known coordinates and measurements of distance Di from $(\bar{x}, \bar{y})$ to these points [6]. A node can estimate its position given three or more reference points in range. The accuracy of the estimate is limited by the precision of the distance measurements and the reference point location accuracy. The measurements could be based on RSS, TOA, TDOA or AOA information.

Triangulation can result in various patterns of circles. The patterns can be classified depending on the number of solutions to the system of equations [7]. The most common patterns of circles are shown in Figure 2.1. Each circle is centered at the location of the corresponding beacon and its radius corresponds to the perceived distance which in our case is approximated by the signal strength sensed by the unknown node. The points in the

intersection represent the possible locations of the unknown node after solving the set of equations.

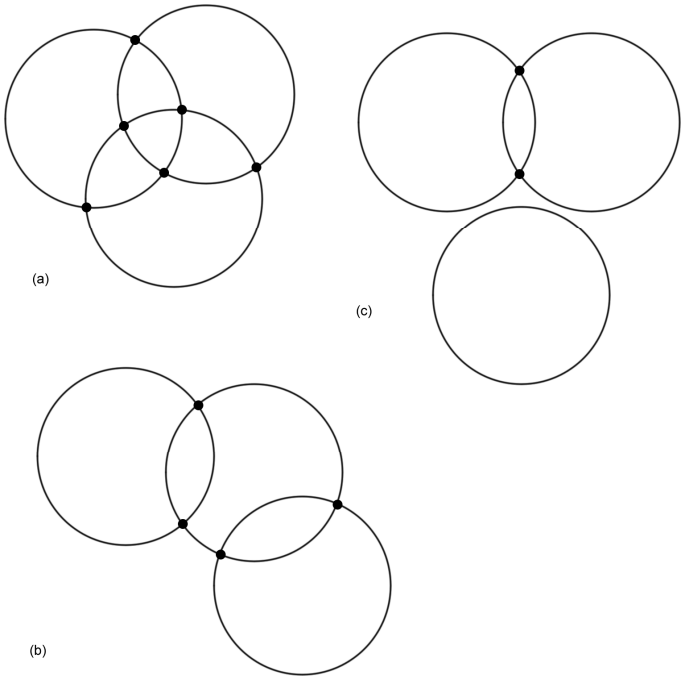

Figure 2.1 Common circle patterns. (a) Type 1 (b) Type 2 (c) Type 3

As shown by [7], Type 1 has six solutions, Type 2 has four solutions, and Type 3 has two solutions.

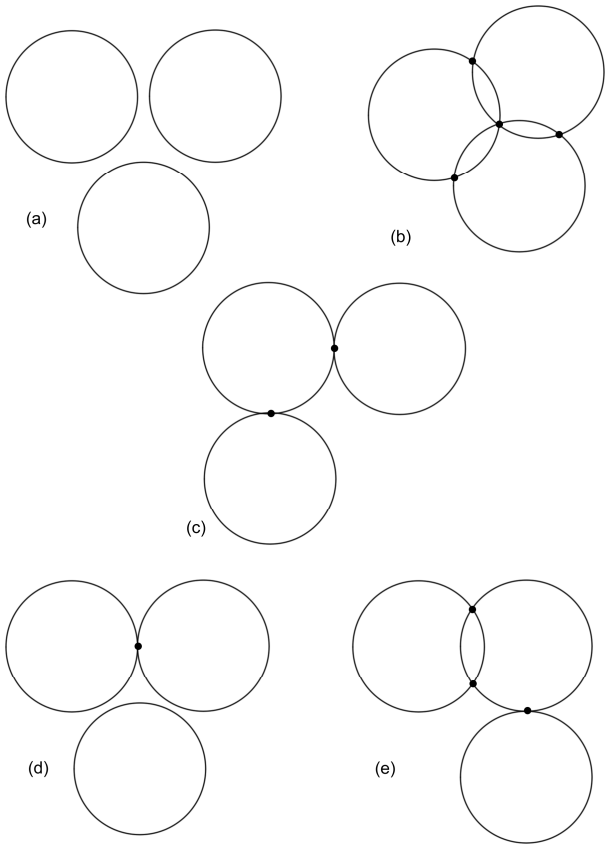

Figure 2.2 Uncommon circle patterns

Some less common circle pattern are shown in Figure 2.2. Figure 2.2 (a) shows a pattern with no solutions, (b) shows the most desirable pattern, the triple root point, (c) two double roots, (d) one double root, and (e) two real solutions and one double root. In any of these patterns, it is possible to determine the location of a node when at least one solution exists. If two or more solutions are found we can decide its location by calculating the centroid of all possible solutions, that is the point located at an equal distance from all the solutions.

2.2.2 Hyperbolic Trilateration

Hyperbolic Trilateration is a geometric method to solve location problems. It works by calculating the intersection point of three circles. The method is mathematically equivalent to triangulation; with the difference that trilateration only works with distances and not angles. There are several hyperbolic trilateration schemes for node location on scattered networks, Atomic Multilateration, Iterative Multilateration, and Collaborative Multilateration. Atomic multilateration covers the basic case where an unknown node can estimate its location if it is within range of three or more beacons. Iterative multilateration is the name given to atomic multilateration when used in the context of ad-hoc networks. It is a distributed algorithm that works by first determining the location of unknown nodes using atomic multilateration. It chooses the unknown node to be closer to the most beacons possible for better accuracy and faster convergence [8]. After determining its location, the node converts into a beacon for other unknown nodes. The major drawback to this algorithm is the error accumulation that results from the use of unknown nodes as beacons. Collaborative multilateration occurs when some unknown nodes do not meet the conditions for atomic multilateration. When it occurs, the unknown node may be able to estimate its position by location information over multiple hops [9]. To estimate its location, sufficient information must be available to solve the system of quadratic equations.

2.2.3 Cooperative Location Topologies

Cooperative location algorithms can locate nodes in the network that are various hops away from location aware beacons. The location topology refers to where the calculations are

determined; be it on a central node or in a distributed fashion. The algorithms discussed in this section take different approaches including a hybrid distributed location topology where some of the calculations are determined locally, and final calculations are determined at a central node.

2.2.3.1 Centralized

Centralized topologies work by taking measurements at the nodes and sending all this information to a central server where location is determined. This kind of scheme usually suffers from relatively large amounts of communication overhead if only a small fraction of the network nodes are in range of location aware beacons. It might be useful for locating nodes in range that have strict constrains on processing power. In other cases a distributed topology is more appropriate for location algorithms [10, 11].

2.2.3.2 Distributed

Distributed topologies are needed when the network has limited communication bandwidth and lacks a central processing node. The processing power is distributed locally among neighboring nodes and location information is spread trough the network starting at location aware beacons. No matter what algorithm is used, the minimum amount of beacons needed to localize most of the nodes in the network is three. All algorithms based on network/cooperative/distributed multilateration suffer from a greater amount of error accumulation than atomic multilateration. Successive refinement is a method used to minimize the error accumulation impact upon location estimates. Also hybrid methods could reduce errors by dividing the network in smaller computational clusters that reduce both communication overhead and error accumulation, and then a central node merges and

optimizes local estimates. We now discuss example algorithms that implement distributed location.

2.2.4 Ranging

Weidong et al. [12] present a range-free geometric localization technique that adjusts the beacons' radio range to localize unknown nodes. Their objective is to obtain higher location accuracy estimation while taking less communications overhead with a small number of anchors. The method starts with two beacons iteratively producing a series of ring intersections and narrowing down the possible area in which the sensor node resides. It calculates the centroid of the intersection area as its estimated location

The algorithm can be described in three steps. First, the beacons broadcast their location information and transmitting power. The unknown nodes that receive the beacon store the information. If an unknown node hears from two or more beacons, then it should select only two of them as its reference beacons. After the reference beacons detect the unknown node, they proceed to gradually decrease their transmitting signal strength and judge whether the unknown node is still covered. The second step is to determine the area formed by the intersection points. After lowering to a minimum range, the unknown node could be located in one of four different cases. Depending on the case, an additional beacon might be needed. In the third step a centroid is calculated to estimate the highest probability location area and thus its final estimated location.

2.3 Related Technologies: Moving Objects Databases

Moving objects databases manage the information of transient location objects. Our system produces this kind of information, the information about the current location of moving objects. All this information could be stored into a database for the usual purposes of current location, or it could be stored into a moving object database. These kinds of special databases are capable of managing different types of specialized queries. Queries may pertain to the future or the past, and triggers are queries related to events [14]. Wolfson [14] gives some examples of queries and triggers. An example of a past query could be “during the past year, how many times was X late by more than 10 minutes at some station”. A trigger could be exemplified by “send a message when a helicopter is in a given geographic area”. A future query might be “retrieve the trucks that will reach their destination within the next 20 minutes.

The importance of moving objects databases is that they address several drawbacks to point-location management, a term that could be applied to an RTLS. Point-location management does not necessarily enable interpolation or extrapolation on all the collected information. Another problem of the point-location method is that it leads to a precision/resource trade-off that can become critical. To increase accuracy, updates must be performed at increased rates, thus sacrificing resources such as bandwidth and processing power.

3 RTL INFRASTRUCTURE

In this chapter we present the design of our real time location system infrastructure. We start by presenting the hardware components and technologies used on our design and the reasons we selected them. This could be considered part of the implementation rather than design, but the design relies heavily on the hardware to be used. We are discussing only the most relevant features of the technologies used.

Fist we present the selected transceivers, microcontrollers, and the network protocol. Then we discuss some issues concerning the distance measurement method. Finally we discuss the centralized real time location scheme.

3.1 Design Requirements

We start by defining the requirements and then looking for solutions. The main goal is a real time location system based on a wireless mesh network. The requirements for this system are:

- Low-cost
- Low-power
- Mesh networking
- Centralized location server
- Small footprint node hardware
- Limited computing capacity at the nodes to keep cost low

- Limited storage capacity at the nodes to keep cost low
- Use the signal strength as a measurement of distance
- Use standardized technologies

Other requirements were drawn upon several iterations of the design and a specific application as the motivation. Figure 3.1 depicts a block diagram of the system. Several nodes are scattered trough the network and some are directly connected to the central server. No central communications antenna is needed in the network. The central server gathers RSS data from all the nodes and uses it to computes the approximate location for each one.

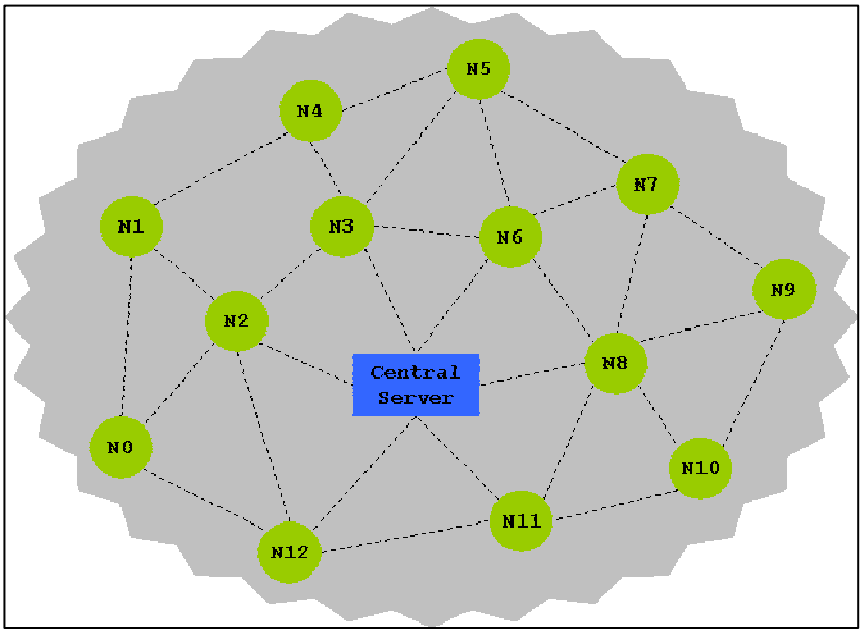

Figure 3.1 Network Block Diagram

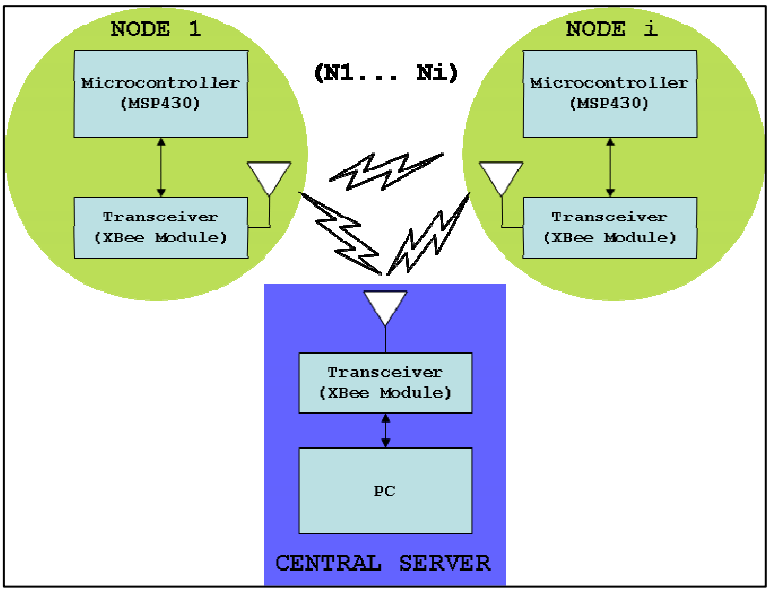

Figure 3.2 Modules Block Diagram

Each wireless node is composed of a wireless transceiver and a processing microcontroller as shown in Figure 3.2. A microcontroller is used since we do not need great amounts of processing power and storage capacity. The central server does need greater processing power and storage since it performs more complex calculations using large amounts of data, thus it is composed of a wireless transceiver attached to a personal computer or server. Based on the requirements that we have laid out for our application domain we have selected to combine Digi's XBee wireless module as the wireless transceiver and Texas Instrument's MSP430 microcontroller for the wireless nodes.

3.1.1 Digi XBee Wireless Module

The XBee wireless module is a stand-alone, ready-to-use device designed for low-power, low-cost, wireless sensor networks. The modules are controlled via a serial interface

and a predefined API command set. The API is used to perform configurations on the module, to control the hardware on the XBee and to perform several networking tasks. These modules operate within the ZigBee mesh networking protocol at the network level or MAC level. The network could grow up to 65,000 wireless nodes with unique addresses. The network also supports point-to-point, point-to-multipoint and peer-to-peer topologies. The modules have an indoor range of up to three hundred feet and outdoors of up to one mile. Our design uses the modules as wireless transceivers with mesh networking capabilities.

Additionally these modules have the capability of sensing the RSS of received data frames. This information is available trough a pin on the module which outputs a pulse width modulated signal (PWM). This signal is used internally by the module as part of its ZigBee routing algorithm. It is only available via this pin or at the MAC level firmware. We are coupling a microcontroller to add more functionality and control to the module, and to gain access to this RSS information.

Our network consists of one coordinator node which is attached to the central server and up to 64,999 batteries operated wireless nodes.

3.1.2 TI MSP430F1232 MCU

As stated above, we need a microcontroller to add functionality, control, and access the RSS information at the network level. We selected the Texas Instrument's MSP430F1232 microcontroller. The MSP430 is an ultra low-power microcontroller and consists of several devices featuring different sets of peripherals targeted for various applications. Its architecture, combined with five low power modes is optimized to achieve extended battery

life in portable measurement applications. It features a 16-bit RISC CPU, 16-bit registers, a digitally controlled oscillator, 16-bit timer, 10-bit A/D converter with integrated reference and data transfer controller and twenty-two I/O pins. In addition, the MSP430F1232 microcontroller has built-in communication capability using asynchronous (UART) and synchronous protocols.

Any other ultra low-power microcontroller with similar capabilities could be used. In our design we are using the MSP430's UART module for asynchronous serial communication with the XBee module. The microcontroller talks with the module to send and receive data frames and to change the modules configuration. We are also using its basic clock module, timer, and capture/compare module to read the RSS PWM signal.

3.1.3 Hardware Interface

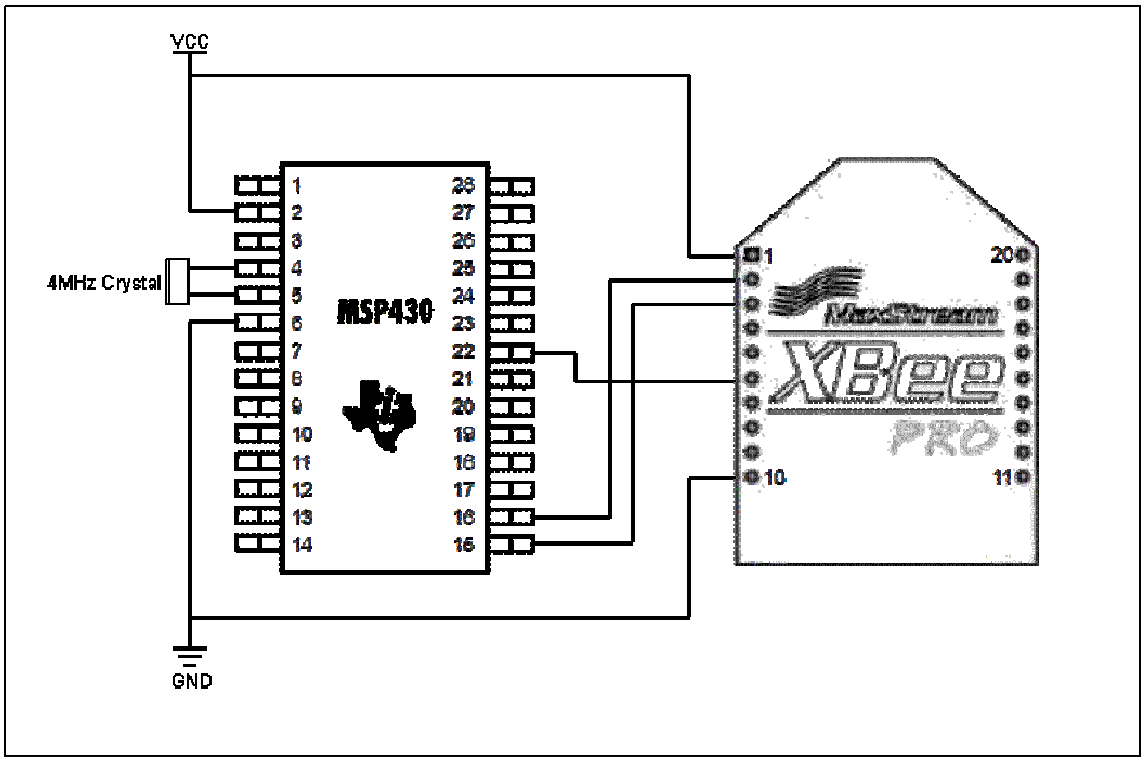

Figure 3.3 Hardware Block Diagram

Figure 3.3 shows the hardware interface for the wireless nodes. It is a simple configuration with minimal interconnection and hardware requirements. Both devices, the XBee and the MSP, share the same power source since they both work at 3.3V CMOS logic. The additional interconnections are the UART serial port and the RSS pin. The resulting device might not be much larger than the XBee module including a coin sized battery. Figure 3.4 shows the interface between the coordinator node and the central server. The XBee modules can be interfaced to a PC via an RS-232 serial connection or a USB-to-Serial converter module.

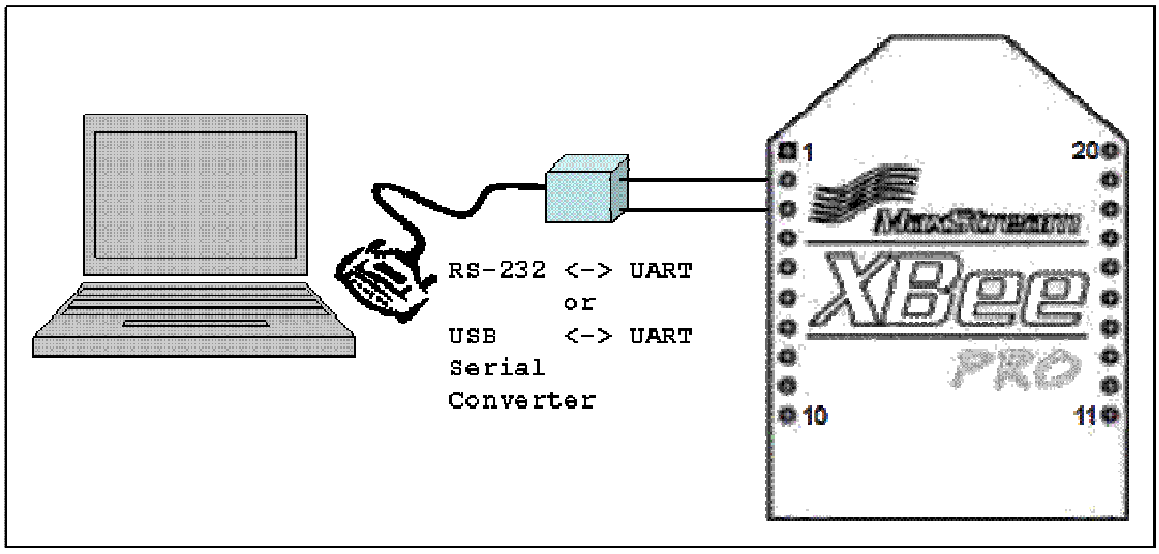

Figure 3.4 PC – Coordinator Interface

3.2 ZigBee Mesh Networking Protocol

The biggest advantage of mesh networks is their ease of deployment, maintenance and scalability. We have selected the ZigBee mesh networking protocol [16] as our communications standard. ZigBee is a network layer protocol that uses the MAC layer IEEE 802.15.4 standard as a baseline. It was developed by the ZigBee Alliance, a group of companies that worked in cooperation to develop a network protocol to be used in a variety of commercial and industrial low data rate, low power, and low cost applications. It adds mesh networking to the underlying 802.15.4 radio. The radios would automatically form a network without user intervention. ZigBee also has the ability to self-heal the network. If a radio at a mid point is removed for some reason, a new path would be used to route messages. This behavior is shown in Figure 3.5.

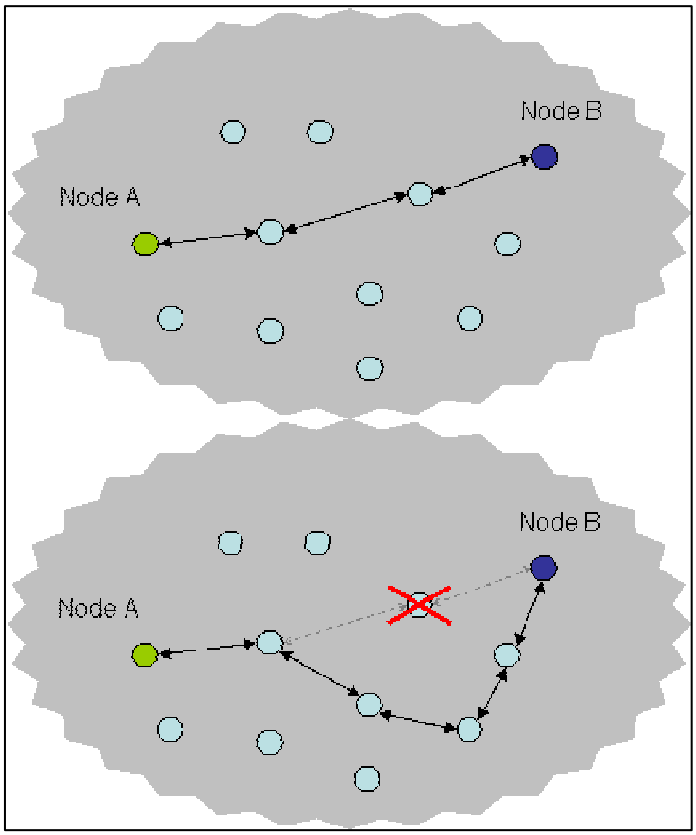

Figure 3.5 Self Healing Mesh Network

3.3 RSS Indicator

As stated before, the received signal strength (RSS) can be used as a metric of distance from node to node. The loss on the signal can be used as a proxy of the distance between the measuring nodes. We will use it since it lowers the cost of our device, it simplifies the development and it is available on most devices including the XBee.

The greatest problem with RSS is that it suffers inaccuracies caused by different sources of error. RF signals decay proportionally to the distance squared (d^2) on free space

and clear line of sight (LoS). This is normal and desirable; but the environment variables cause two major sources of error for RSS that are shadowing and multipath signals [2]. Multipath signals are caused when there is an obstructed LoS and objects scatter RF signals on different paths. These multipath signals arrive at the receiver with different amplitudes and phases, adding or constructively or destructively as a function of the frequency, causing frequency selective fading [1]. A solution to this problem could be to average several measurements of a single frequency over time or to use a spread-spectrum method to average the received power over a wide range of frequencies. Using a wideband method to measure the power of the received signal is equivalent to measuring the sum of the powers of each multipath signal.

Shadowing is the attenuation of the signal due to obstructions along the path that the signal must pass trough or diffraction around the object. These are typically considered random error sources. Other sources of error could be attributed to measurement circuitry precision and calibration but are usually considered insignificant.

RSS errors are considered to be multiplicative, in comparison to other techniques where error sources are considered to be additive. Thus RSS is considered to be better suited to high density sensor networks.

Given the scope of this research, we are not focusing our efforts on improving the RSS indicator quality and accuracy. Instead we can characterize the RSS to distance relation for a given environment setting. We hypothesize that this method should yields acceptable results.

3.4 Centralized Location

A mesh networking is by definition self configurable and self healing. Our distributed location scheme takes advantage of these characteristics to reduce the costs of location in a network of wireless nodes. A mesh network does not need a wide coverage antenna since each node routes trough its surrounding nodes, thus reducing hardware costs. Another way of reducing costs is to minimize the amount of fixed location nodes. Each node in the network might be located by taking advantage of a GPS device, yet this increases the cost and size of the devices. Our approach uses at least three nodes that serve as initial beacons. The beacons must have a known location. Using cooperative multilateration most nodes in the network could be located within an acceptable margin of error for many applications.

Cooperative multilateration works by locating the unknown nodes closer to at least three beacons. This is accomplished by trigonometric multilateration using the distance among nodes as measured by the received signal strength. Their location is then propagated to the farther unknown nodes by using the location of the newly located nodes. Figure 3.6 shows the way location spreads. We start with a) three beacon nodes, shown in blue, and several unknown nodes in range, shown in green. After the first iteration we can b) locate farther unknown nodes using the newly found nodes, shown in black. This method continues until c) most nodes are located within the network. Nodes that are not reached by three or more nodes d) cannot be located with precision.

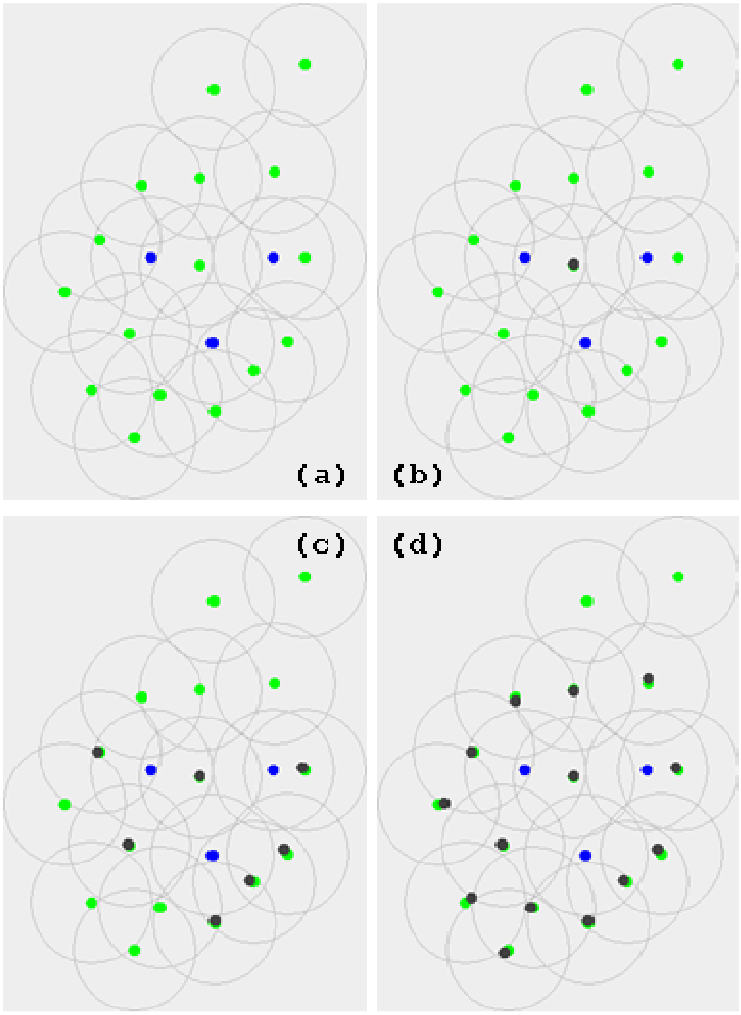

Figure 3.6 Location Propagation

The calculations of this cooperative multilateration could be achieved either at the node or at a centralized server. Our approach uses a centralized server for several reasons. Most of our target applications are related to centralized monitoring, so there is no need for the nodes to know their location. Another reason is that we can do better data filtering at a centralized level than at the nodes. The server has information about additional nodes that can help improve location precision. A node can do data filtering related to the nodes that are

connected to it, but not the ones that are relatively close but not reachable. The reason for this is that the nodes have limited computing, communication, and storage capabilities to keep them under design requirements.

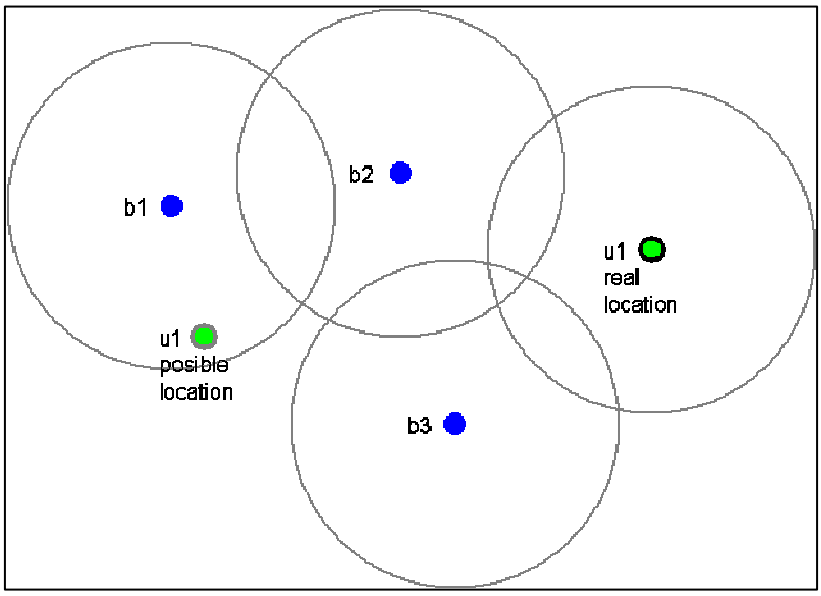

Figure 3.7 Special Case

Figure 3.7 shows a case where we can make a good prediction of an unknown node `u1` at a centralized server, but not at the node level. The central server has data about the location of all beacon nodes (`b1, b2, b3`), but if unknown node `u1` only has data of `b2` and `b3`, it is impossible to choose the correct solution. Yet it is still possible for a node to gather the necessary data to achieve the same level of knowledge as a server, but this will increase the data transfer on the network as well as the storage capacity requirement for the node which in or case should be kept minimal.

An interesting methodology, not included in our discussion, would be the utilization of a mobile beacon as introduced by [13], but in the context of cooperative multilateration. This mobile beacon could reduce the impact of error accumulation while minimizing the cost of the system as a whole. We leave the implementation of this modification for future work.

4 RTL ALGORITHMS

This section describes in detail our implementation of a real time location system for wireless mesh networks. We start by describing the system as a whole. Then we describe the wireless nodes behavior and internal algorithms. We then introduce the central server data gathering, trilateration and filtration algorithms. We also describe the data flow in the network of nodes. Finally we will explain the relationship between the RSS and the distance among nodes.

4.1 Trilateration Algorithm

As stated in chapter 2, trilateration is a geometric method for solving location problems. It could be used to estimate two dimensional locations as well as three dimensional locations. Our systems will be based on trilateration since our data is appropriate for the algorithm. We gather RSS data which could be translated to distance information useful to the algorithms. The trilateration algorithm is very simple. It works by calculating the intersection point of three circles. The method is mathematically equivalent to triangulation; with the difference that trilateration only works with distances and not angles.

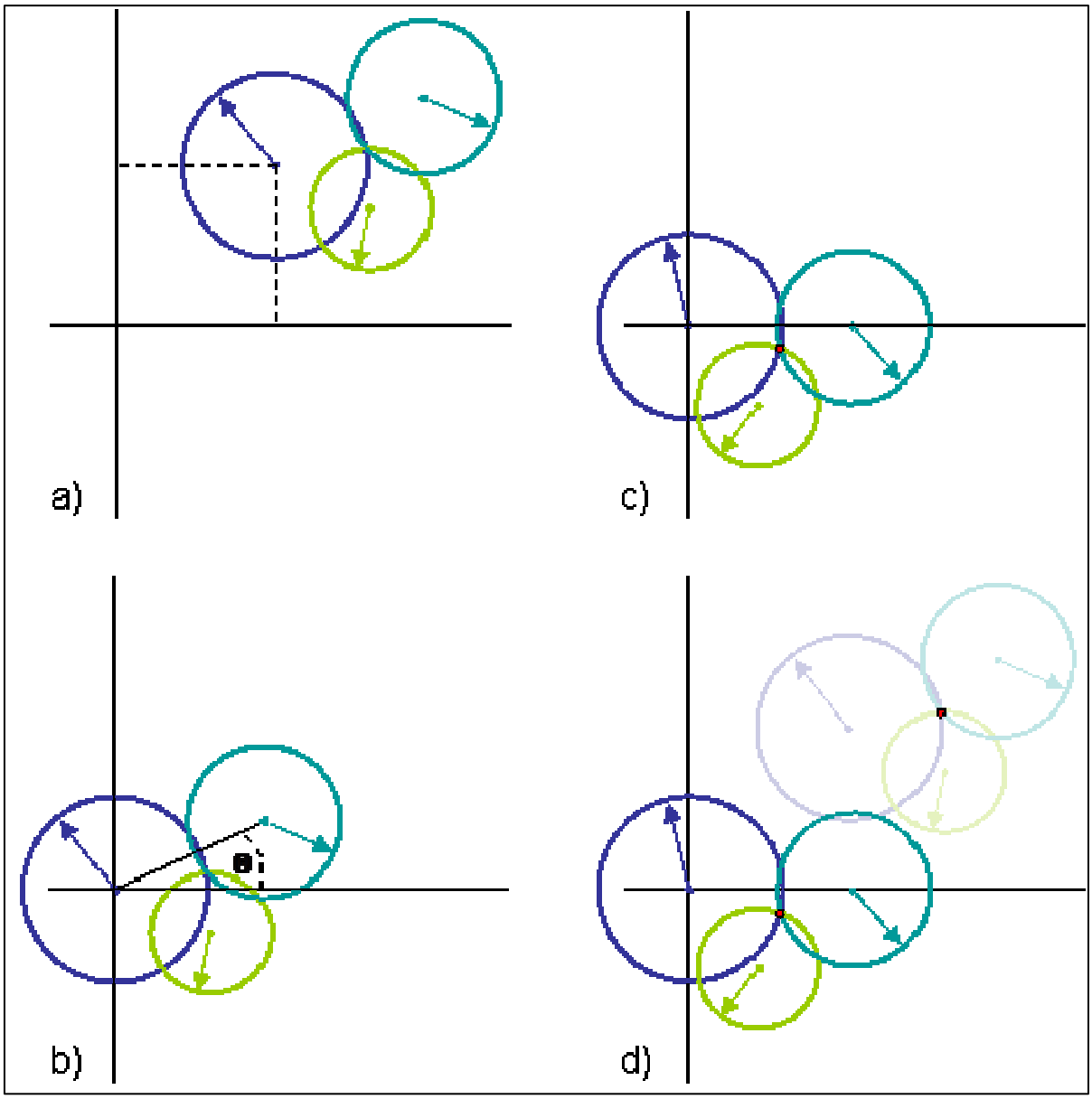

Figure 4.1 Trilateration

Figure 4.1.a shows three circles intersecting at a single point. Our goal is to find the coordinate where all three circles are intersect, that being our unknown node location. To make calculations simpler, we will choose one of the circles to be at the origin and another to

be at the x axis. The third circle will be called *beacon* and the unknown point will be called *unknown*.

```
origin = circle at origin
xAxis = circle at x axis
beacon = third circle
unknown = unknown point
```

After selecting the origin circle, all three circles are translated as shown in Figure 4.1.b using the following calculations.

```
tOx = 0
tOy = 0
tXx = xAxis.X - origin.X
tXy = xAxis.Y - origin.Y
tBx = beacon.X - origin.X
tBy = beacon.Y - origin.Y
```

The circles are then rotated by theta Θ, to place the xAxis circle on the x axis. This is shown in Figure 4.1.c and is accomplished as follows:

$$\Theta = \tan\left(tXy \middle/ tXx\right) \quad \textbf{4.1}$$

```
ttOx = 0
ttOy = 0
```

$$ttXx = \sqrt{tXx^2 + tXy^2} \quad \textbf{4.2}$$

```
ttXy = 0
```

$$ttBx = tBx \times \cos(\Theta) + tBy \times \sin(\Theta) \quad \textbf{4.3}$$

$$ttBy = tBx \times \cos(\Theta) - tBy \times \sin(\Theta) \quad \textbf{4.4}$$

Having all points in place we can compute the unknown location by equating all circles to find the coordinate where all three intersect.

```
origin = xAxis = beacon
```

Where ,

```
rO = origin radius
rX = xAxis radius
rB = beacon radius
ttUx = unknown x coordinate
ttUy = unknown y coordinate
```

and where each circles equation is given by,

$$rO^2 = ttUx^2 + ttUy^2 \quad \textbf{4.5}$$

$$rX^2 = (ttUx - ttXx)^2 + ttUy^2 \quad \textbf{4.6}$$

$$rB^2 = (ttUx - ttBx)^2 + (ttUy - ttBy)^2 \quad \textbf{4.7}$$

To solve the equations for `x`, we can subtract the `xAxis`'s equation to `origin`'s equation.

$$ttUx = \frac{rO^2 - rX^2 + ttXx^2}{2 \times ttXx} \quad \textbf{4.8}$$

To solve for `y` we substitute back on `origins`'s equation, then we equal this resulting formula to the `beacon`'s formula and solve for `y`.

$$ttUy = \frac{rO^2 - rB^2 + ttBx^2 + ttBy^2}{2 \times ttBy} - \frac{ttBx}{ttBy} ttUx \quad \textbf{4.9}$$

Now we have found our unknown point. The remaining calculations are to rotate back by $-\Theta$ and translate by the distance of the original `origin` circle, as shown in Figure 4.1.d.

4.2 RSS to Distance Relation Characterization

We have already defined RSS as the sensed or measured power metric of the last received transmission by a node. We have also stated the reasons for selecting RSS as our

measurement of distance between nodes, being those price, simplicity and availability. This section will be devoted to describe the way we translate the RSS PWM signal available on the XBee module to an approximate measurement of distance.

The XBee's documentation [1] states that it internally measures the power of the signal of the last RF frame received by the module. The method used to capture the power is neither described nor important for us. They do describe the signal as a pulse width modulated signal proportional to the received power at the module. It is a sixty four nanoseconds square wave. The percentage of a high voltage to low voltage of this period will translate to the fade margin of the radio. For example if our PWM signal has a ten percent duty cycle, it translates to a ten decibels fade margin. The XBee's documentation [1] does not mention the XBee's resolution for that measurement, but their support staff has stated that "the period is 64 microseconds and there are 445 steps in the PWM output. So the minimum step size is 144 nanoseconds". This means that the PWM signal has a resolution of 0.225 dBm (decibels milliwatt). The receiver sensitivity is given by the documentation as -100 dBm.

All this information could be useful with the help of a radio propagation model. This model is an empirical mathematical formulation for the characterization of the radio wave propagation as a function of distance and other conditions. We do not have a defined path loss model for the XBee modules nor do we try to define one since it is out of the scope of this research. Instead we can characterize a model using experimental data for a given environment. This might affect the precision of the location algorithm since a scientifically developed model could yield better approximations.

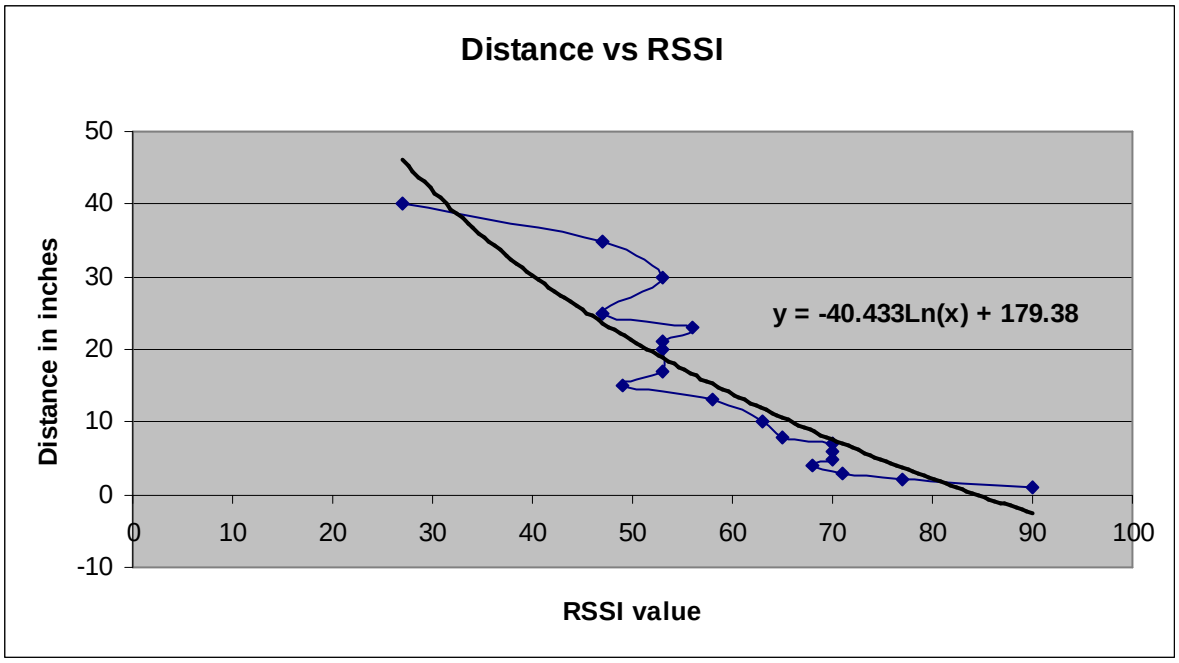

Figure 4.2 Distance vs. RSS characterization

Figure 4.1 shows the characterization of our experimental setting. In this scenario we have two XBee-Pro modules transmitting at power level 1 (12 dBm), with no attached antenna and inside a 20' x 20' concrete walls room. The XBee-PRO module has a range of five power output levels from 10 dBm to 18 dBm. The RSS measurements were done with an MSP430F1232 microcontroller using its internal timer and capture/compare module. The MSP430F1232 capture algorithm is described in the next section.

In this scenario we have found a similar logarithmic function that relates to an RSS value and a distance. Notice that our setting is only experimental since the modules do not have antennas and are not working at their maximum power output. As specified by the manufacturer, the indoor/urban range of the XBee-PRO modules is up to 300 feet and the outdoor line-of-sight is up to 1 mile. The modules were not designed to work without

antennas, so, the erratic behavior in the range of 25 to 40 inches does improve with one attached. We used them without antennas for this experimental setting. By limiting the range to up to 40 inches we can develop the experiments with a greater control over the measurements.

4.3 Wireless Nodes Firmware

This section will briefly explain the software developed to run on all the wireless node microcontrollers and their behavior. The code has been simplified to clearly present the algorithms. The nodes have several tasks to do besides creating and maintaining the wireless mesh network. Each node receives and performs commands sent by the central server, and also returns RSS values to the central server and propagate server broadcasts.

4.3.1 API Mode

XBee modules have two modes of operation, command mode and API mode. We will only describe one, the API mode. This mode specifies the application level protocol in which modules and devices talk to each other, be it wirelessly or via serial port. Figure 4.3 shows the structure of the API data frames. Any module can communicate with another module in the network by defining the destination address and other predefined commands. The remaining details are not important in our discussion.

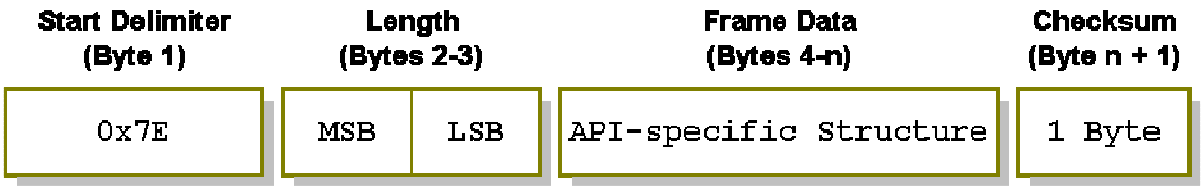

Figure 4.3 General XBee API Frame Structure

4.3.2 Wireless Node Main Algorithm

Each node in the network runs the same algorithm inside their microcontroller. The algorithm is summarized as follows and later described in detail.

- At start up, the microcontroller sets up all the internal modules and timers, and goes into low power mode to wait for an incoming data frame from the transceiver.
- When a data frame is received it is parsed to extract information and commands.
- If the frame contains a command it is performed in the transceiver.
- If the frame contains a broadcast from the server, the broadcast id is looked up in the internal database.
- If the broadcast id is found, the RSS value is determined and sent back to the central node. The loop starts again.
- If the broadcast id is not found, its id is stored and the frame is broadcasted again to nearby nodes. Finally the RSS value is determined and sent back to the central node. The loop starts again.

The central server sends from time to time a broadcast frame to all nodes. This broadcast frame serves as a signal for RSS sensing and value return. Figure 4.4 shows the structure of this frame.

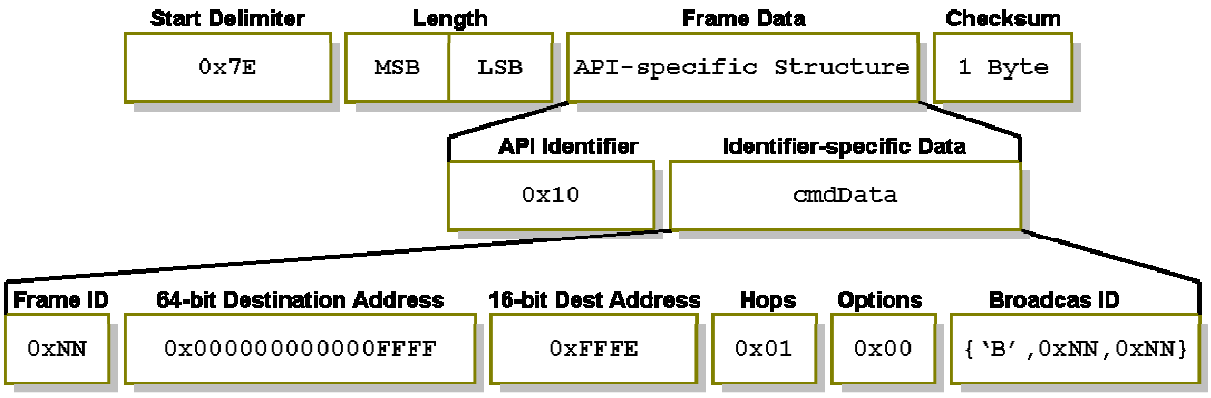

Figure 4.4 Broadcast Frame

Notice the destination address and maximum hops count. The destination address specifies the broadcast address and up to one hop count. This means that each frame spreads only to directly linked nodes. This behavior ensures that nodes will receive a frame from each physical link. All RSS data generated this way is sent back to the central server. It is there where the collaboration among nodes is observed, since all nodes worked together to determine their physical links and to route the data to the destination server. The microcontroller is in charge of keeping a list of recent broadcast frames for re-broadcast. This is necessary to keep track of the sender address. It changes with every re-broadcast.

```
main(){
  // Loop for ever
  for(;;){
    // Go into low power mode and wait for an interrupt
    goIntoLowPowerMode();
    // An event has occurred and returned from low power mode
    // A broadcast frame was received?
    if (isTxReady()){
      // Sense RSS value for this last frame
      senseRSS();
      // Prepare RSS return frame
      returnRSSFrame[];
      // Send RSS return frame to central server
      sendFrame(returnRSSFrame);
    }
    // Do a re-broadcast?
    if (ifRetransmit()){
      // Prepare Broadcast frame
      reBroadcastFrame[];
      // Send Broadcast
      sendFrame(reBroadcastFrame);
    }
    // Execute a command?
    if (ifCommand()){
      // Prepare Command frame
      commandFrame[];
      // Execute Command on module
      sendFrame(commandFrame);
    }
  }
}

UART_RX_INTERRUPT(char receivedChar){
  // Parse received byte
  digest(receivedChar);
  // Complete frame received?
  if (completeFrame()){
    // Broadcast frame?
    if (isBroadcast()){
      // Retreive sender address and broadcast frame ID
      senderAddress64[];
      frameID[];
    }
    // Command frame?
    else if (isCommand()){
      // Save command ID and value
      commandID[];
      commandValue[];
    }
  // Exit Low Power Mode after return from interrupt
  clearLowPowerMode();
  }
}
```

Figure 4.5 Wireless Node main algorithm pseudo-code

Figure 4.5 shows the pseudo-code for part of the internal algorithm executed by the microcontroller. We do not have an operating system running on the microcontrollers. Thus,

events are managed by hardware or software interrupts. The MSP430's UART module generates an event for each character received. This event halts the execution of the main code. Events are also generated at the timer capture/compare module. The microcontroller is usually in sleep mode to save power.

Let's start the description at the UART interrupt function, `UART_RX_INTERRUPT(char receivedChar)`. This interrupt function has been simplified from the actual function and algorithm. As previously stated, it is called via a hardware interrupt every time the UART module receives a character. This character is accumulated until a complete and valid frame is received. Once a valid frame has arrived, important data is extracted and saved for later use. There are two types of frames that are relevant for us, broadcast and command. When a broadcast frame is received, the sender address and the frame ID are stored. For a command frame, the command ID and the command value are stored. Nothing else besides data extraction is done at the interrupt service routine. Data flows from the XBee module at random intervals; therefore we must reduce the complexity of the interrupt routine to a minimum.

The `main()` loop manages the most important logic. As shown in Figure 4.5, execution stops until an event occurs. The microcontroller exits low power mode when an interrupt occurs and the mode is cleared inside the interrupt service routine. Going back to the interrupt service routine, the low power mode is cleared only when a complete and valid frame is received. After receiving a frame, the main loop awakens and continues execution. It starts by verifying if a broadcast was received in which case the RSS must be determined and sent back to the central server. This process will be described in more detail in the next

section. After this step is completed, is the node determines if the broadcast was previously received. If it was not received, the broadcast ID is stored in flash memory and resent to the closest nodes. If the frame contained a command, it is executed. Most commands are directed to the XBee module to modify some properties.

4.3.3 RSS Sensing

```
senseRSS(){
  // Start timer and capture/compare module
  // Go into low power mode and wait for 4 captures
  // Other events could occur as well
  if(getCount() < 4){
    goIntoLowPowerMode();
  }
  // Calculate RSS value and return
  calculateRSS();
}

TIMER_INTERRUPT(){
  // Four measurements already?
  if(getCount() < 4){
    // Save current capture
  }
  else{
    // Stop timer
    // Exit Low Power Mode after return from interrupt
    clearLowPowerMode();
  }
```

Figure 4.6 RSS sensing pseudo-code

Figure 4.6 shows the pseudo code for RSS sensing. It has been modified from the original code for simplicity. As revealed from the code, it is extremely simple. First, set the timer and capture/compare module. The setting has been hidden since it is dependent on the device. More important details will be given soon. After starting the timer module, it waits until measurements are complete. Then it does some calculations and returns. Now let's

examine the timer settings. As stated previously, the XBee module outputs the RSS data via a PWM signal.

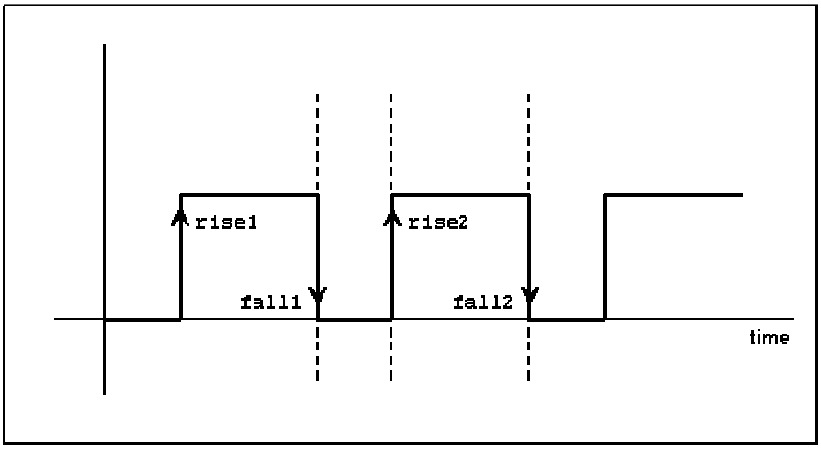

Figure 4.7 PWM RSS signal

Figure 4.7 shows an example of a PWM signal. We need to capture four sequential edge events. In the example given in Figure 4.7, we could read edge events `rise1`, `fall1`, `rise2`, `fall2`. The timer is counting continuously. On each event the capture module registers the timer's value at the triggered event. Having four events registered we can calculate the PWM duty cycle by selecting the first falling edge and then calculating the distance to the next two events. Simple mathematics gives us a duty cycle related to the received signal strength, this is our RSS value. XBee modules use RSS values internally for routing purposes at a network level. We are ascribing additional value to this measurement by using it for our application.

4.3.4 Broadcast Frame ID List

Received broadcasts are re-transmitted only if they are not found on the node's internal ID list; otherwise the network will flood with broadcast frames being sent perpetually. This is a general broadcast algorithm. The list is saved at the MSP430's flash memory. Figure 4.8 shows the simple algorithm that searches and stores these values.

```
boolean received(char[] frameID){
  for (i = 0; i < receivedIndex; i++){
    // Search in blocks of two bytes
    for (j = 0; j < 2; j++){
      if (frameID[j+8]) ==
         *(char *)(0xF000 + j + i * 2)){
        continue;
      }
      else{
        // Not found
        j = -1;
        break;
      }
    }
    if (j == -1){
      // Not found yet
      continue;
    }
    else{
      // Found
      i = -1;
      break;
    }
  }
  // Found broadcast frame ID
  if (i == -1){
    return true;
  }
  // Not found. Add to flash mem
  else{
    // Up to 2KB for keeping frame IDs.
    // 0x07F8 = 2040 ~= 2048 ~= 2KB
    if (currFlashPtr == ((char *)(0xF000 + 0x07F8))){
      // Full buffer. Reset.
      receivedIndex = 0;
      currFlashPtr = (char *)0xF000;
    }
    for (i = 0; i < 2; i++){
      writeToFlash(frameID[i+8]);
    }
    receivedIndex++;
    return false;
  }
}
```

Figure 4.8 Frame ID search and store

It starts by iterating trough the memory segment selected for storing IDs. If the current ID is not found, it stores it in the next available slot. The method `writeToFlash(byte)` is in charge of properly writing to the flash memory module. To minimize a storage capacity the id list is kept in a LIFO buffer.

4.4 Central Server Application

The central server software runs in a personal computer or server since it requires greater power than that of the wireless nodes. It is in charge of managing the coordinator node, configuring the wireless nodes in the network, and calculating the location of all nodes. Location information can be used to draw the location of wireless nodes in a map. Our implementation includes a basic two dimensional map. This user interface will be described briefly on sections ahead.

The central server algorithm and application is summarized as follows and later described in detail.

- At start up the server sends commands to each node in the network to modify or reset some internal parameters.
- The user can configure the server to send broadcasts at time intervals or manually at his/her discretion.
- If the server sends a broadcast frame, it waits for responses from all nodes.
- After receiving a response frame from a node, the server extracts the addresses from the nodes involved and the RSS value for this link.

- After all nodes respond the server iterates trough all its stored data and tries to calculate new locations with the refreshed database value.
- New location data is displayed or sent to a mapping application.
- The process is repeated for each frame received with RSS information.

4.4.1 XBee Software Interface

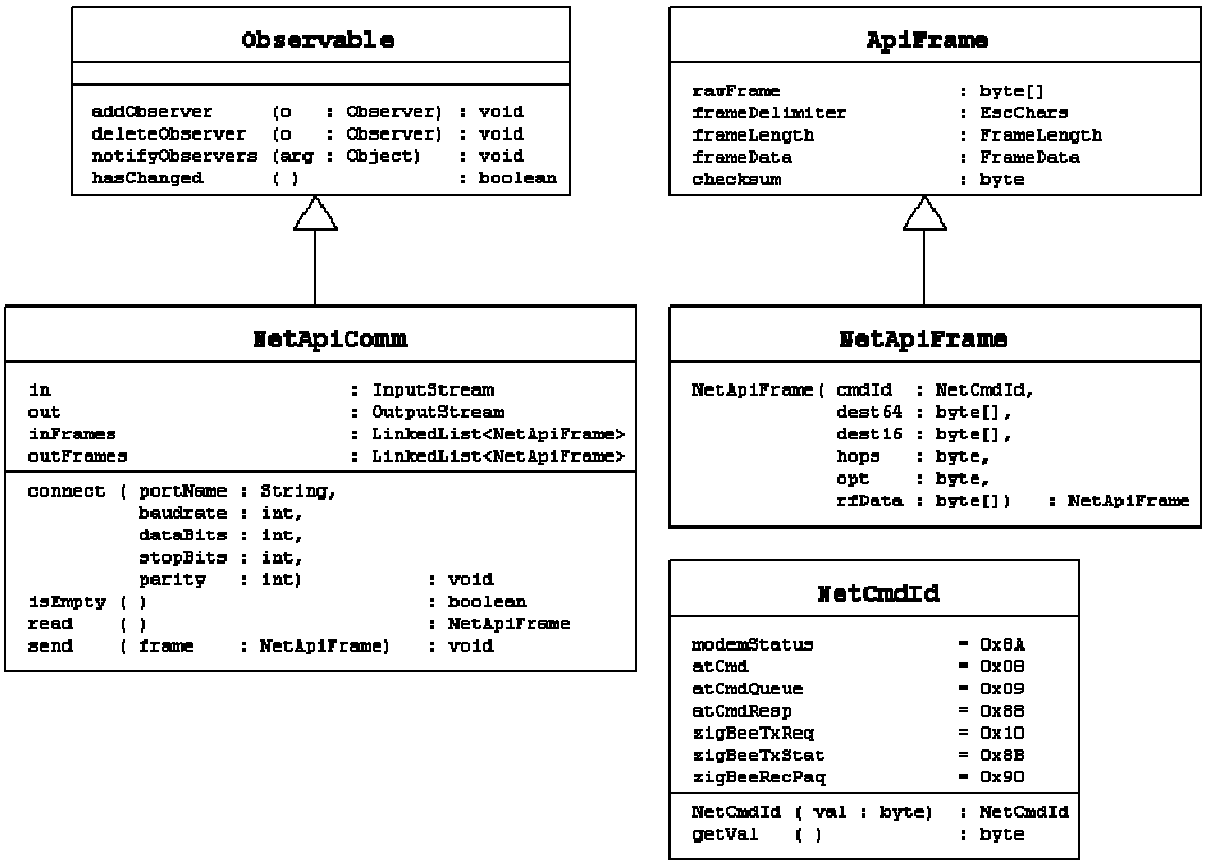

Figure 4.9 Simplified XBee Software Interface Class Diagram

XBee modules where designed to interface via a serial communications channel. They define a proprietary serial communications protocol. The modules have two serial

modes of operation, command mode and API mode. We chose to work with API mode for reliability and compatibility with an object oriented programming model. The modules have a variety of commands to configure the network, configure the module, send and retrieve data and execute some behaviors. We developed a software library in JAVA to easily interface with a module connected via a serial port. Figure 4.9 shows a simplified class diagram of our implementation. `NetApiComm` creates a serial port connection to an XBee device connected to the PC. `NetApiComm` implements two communications paradigms, push and pull. It inherits from the `Observable` class thus observers could join to be notified of incoming data frames. `NetApiComm` also implements several methods to read the next received frame, send a new data frame, and verify if data frames are available. Data frames are created with the `NetApiFrame` class which defines a general frame structure as shown in Figure 4.3. All frame types and command types have been defined and implemented. We also completed a MAC level implementation of the interface, `MacApiComm`, `MacApiFrame` and other related classes. It can be used with the 802.15.4 XBee module's firmware.

4.4.2 Cooperative Trilateration

On section 4.1 we described the mathematical process and formulas for trilateration. This section will focus on the algorithm used by the central server to process all gathered data and actually locate each node on the network.

Let's start with a simple diagram to understand how data flows throughout the network. Figure 4.10 illustrates several time frames of data flowing trough the network. The

coordinator node attached to the main server starts the location broadcast among nodes. Data packets received by wireless nodes could be used to signal the RSS sensing process and the return the information to the central server. The broadcast and return process continues at intervals defined by the server.

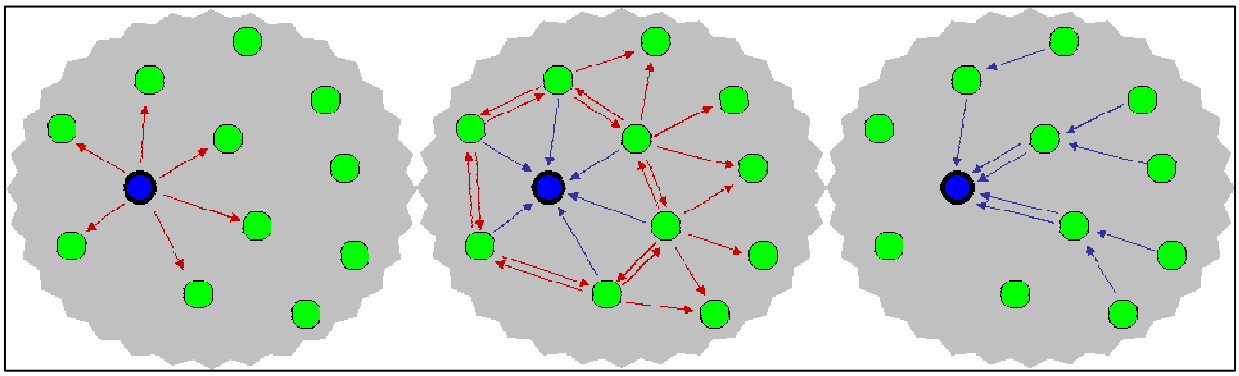

Figure 4.10 Broadcast and Return Data Flow

The coordinator node, shown in blue, starts the broadcast to its neighbors, shown in green. Some neighbors might be beacon (known location) nodes. It is necessary to also know the RSS among beacons to update the graphical user interface. Red arrows represent the spreading of the broadcast following standard broadcasting algorithms. Blue arrows represent the return of RSS data frames for each link. A link is defined as a physical level wireless communication link between two nodes. The links are all the information we need to calculate the locations of the nodes. The central server maintains a database of all the links received. The server iterates throughout all links received to incrementally locate all nodes. Each link contains the IP addresses of the two nodes, the RSS value, a timestamp and a time-time-to-live value.

Figure 4.11 shows a simplified version of the algorithm that runs at the central server. This algorithm performs distributed trilateration. Distributed referring to the fact that location data was gathered by distributed collaboration of nodes and that location information propagates as new nodes are found.

```
void trilaterate(){
  // Iterate to find beacon-to-beacon Links
  Vector<CLink> blinks = new Vector<CLink>();
  // Find their mapped distances
  // Calculate Map-to-Real Ratio
  mtrRatio;
  // Verify there is enough beacons to continue
  if (ls < 3) {return;}
  blinks = null;
  // Iterate to find links with unknown nodes
  for (Iterator<CNode> uit =
       database.getUnknownNodes().iterator();
       uit.hasNext();){
    Vector<CLink> tlinks = new Vector<CLink>();
    /**Filter links**/
    // Find links for current node
    // Remove by duplicates, by timestamp and by unknowns
    // Sort by closest to the unknown, then by beacons
    // At least three known nodes?
    if (bqty < 3){continue;}
    /**Trilaterate**/
    // Select first three filtered beacons
    CNode origin = tlinks.elementAt(0);
    CNode xAxis = tlinks.elementAt(1);
    CNode beacon = tlinks.elementAt(2);
    // Convert with Map-to-Real ratio to get mapping distance
    sigStrO = tlinks.elementAt(0).getRSS() / mtrRatio;
    sigStrX = tlinks.elementAt(1).getRSS() / mtrRatio;
    sigStrB = tlinks.elementAt(2).getRSS() / mtrRatio;
    // Trilaterate
    unknown = Trilateration.trilaterate(
              origin.getLocation(),
              xAxis.getLocation(),
              beacon.getLocation(),
              sigStrO, sigStrX, sigStrB);
    // If no real solution is found return
    if (unknownPoint == null){continue;}
    // Update Map
  }
  // Repeat procedure with known nodes to refresh their position
  for (Iterator<CNode> uit =
       database.getKnownNodes().iterator();
       uit.hasNext();){
    // ...
  }
```

Figure 4.11 Central Server Distributed Trilateration Simplified Algorithm

We start with a database of `CLink` and `CNode` objects. `CLink` is an object corresponding to each physical link gathered from the broadcast and return process. Fixed

location beacons are first retrieved to calculate a Map-to-Real ratio. This ratio is used to convert from real distance to the distance displayed at the user interface. After this, a list of `CNode`‘s containing unknown nodes is retrieved from the database. The algorithm iterates trough all nodes until all possible locations are calculated. For each `CNode` a list of `Clink`s is retrieved from the database. The list contains all physical links to this unknown node. The list then undergoes several filters to clean data from useless data. It first removes duplicate data, and then it removes old data comparing its timestamp to its TTL (time-to-live) value. Finally it removes links with unknown location beacons. The final step towards filtering is to sort links by the closest to the selected unknown, and then sort by fixed location beacons. Step by step, these filters get rid of useless or old data and then sort the list to select the best three known location nodes to perform the trilateration with. After selecting the best three candidates for trilateration, the RSS is converted using the Map-to-Real ratio.

`Trilateration` is an abstract class that implements the trilateration algorithm. The internals of this class are defined and described in section 4.1 and 2.2. After iterating all unknown nodes, the same process is done for previously located nodes to refresh their location.

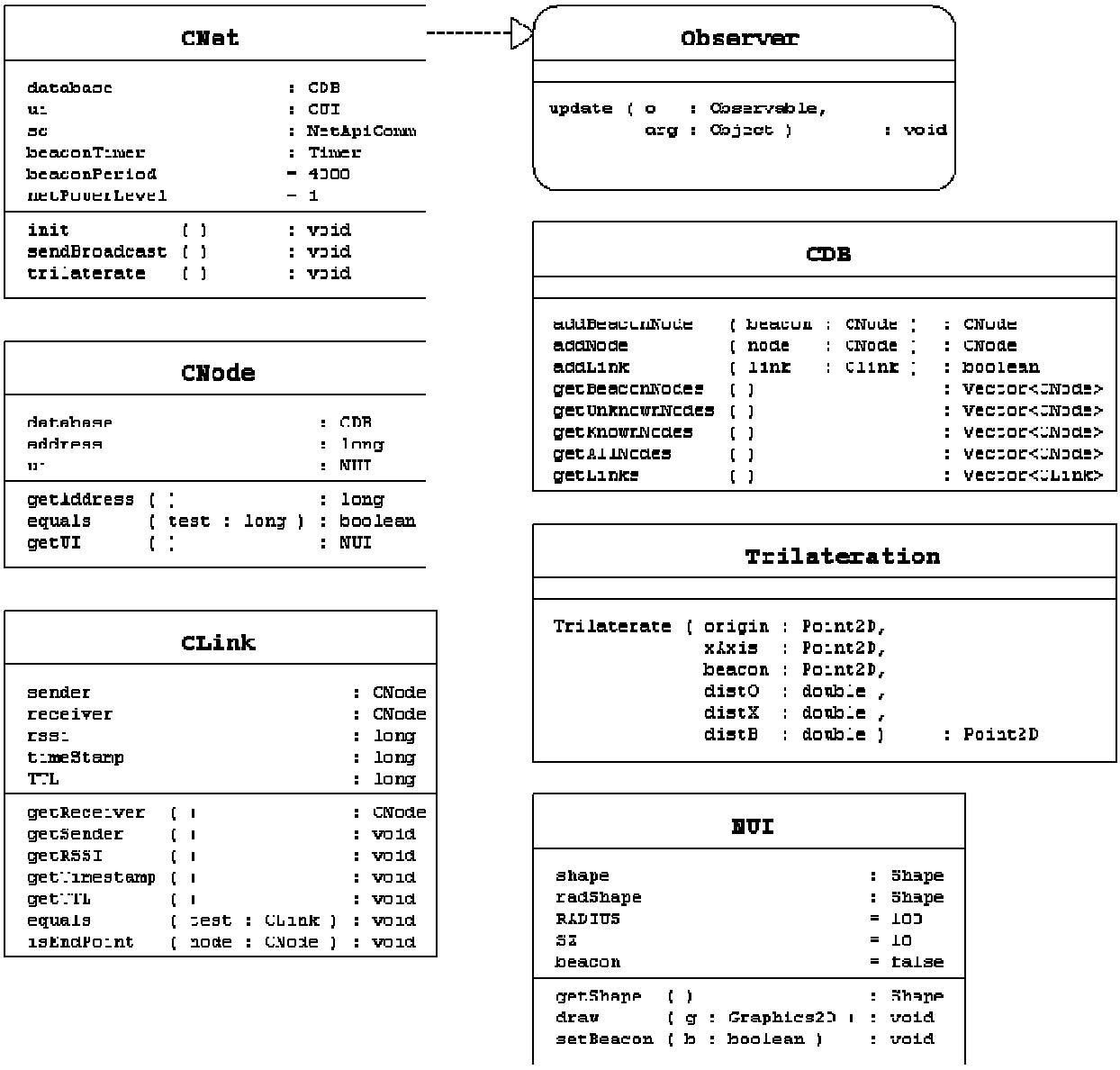

Figure 4.12 Central Server Simplified Class Diagram

Figure 4.12 shows a simplified class diagram for the central server. These classes implement a representation of the real network of nodes and include a graphical user interface for each node. These classes together with the previously described XBee software library comprise the software middleware for real time location systems based on the XBee

platform of wireless sensors. The `CUI` class is not shown in the diagram. This class implements a simple graphical interface. The next section will describe the screens and parts of the graphical user interface.

4.4.3 User Interface

We developed a simple user interface that shows the nodes at the PC screen as they are located. We will not go into class internals and details of this implementation since it is not the scope of our research to develop the graphical interface. The GUI can be customized based on the application. Figure 4.13 displays the system at work. The top half shows the user interface. It maps three beacon nodes arranged in a triangle shape, and a third unknown node. This setting is only experimental, but demonstrates the functionality of the developed software. The bottom half of the figure shows a picture of the actual devices working with the system. Notice that the unknown node was located at a reasonable location at the map, relative to its actual location. This experimental setting only has 4 nodes, but it runs the same software that is able to locate a bigger network of nodes. Another important detail of the graphical user interface is the gray circle surrounding each node. This circle represents the signal coverage of each node. Remember that the wireless modules used do not have an attached antenna and are working at one of their lowest power levels available. We can roughly approximate their range in this scenario to a radius of about 40 inches. We described their range capabilities in section 3.1.1.

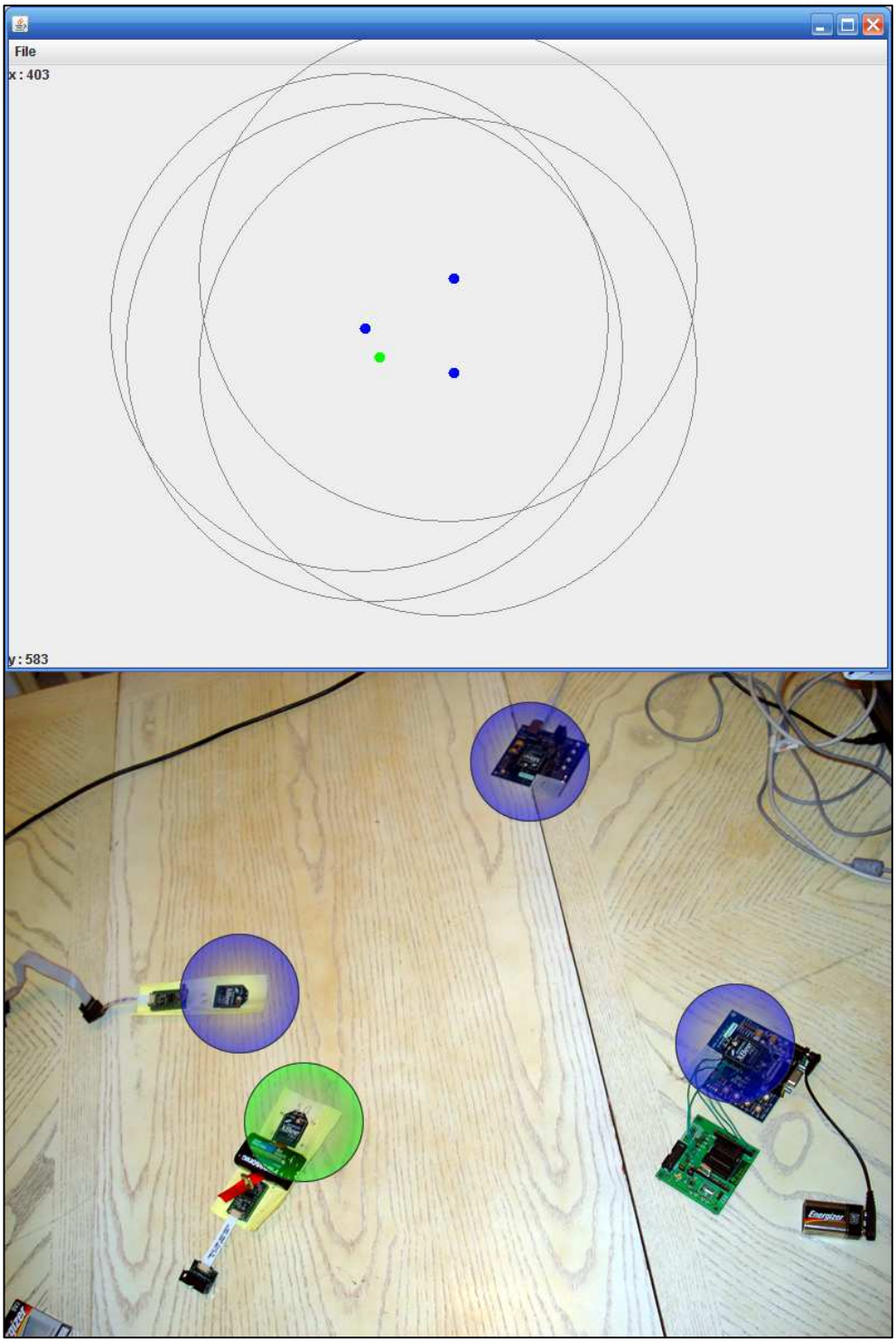

Figure 4.13 M-Loc Prototype

Real-Time Location Systems are inherently dynamic in nature. This dynamic inherent property cannot be observed in Figure 4.13, but our implementation does locate the nodes in real time as they move. It is also out of our scope to analyze and optimize the time delays of the real position versus calculated position or the wireless unknown nodes. Improvements in the microcontroller code and some parameter tweaking at the wireless modules will greatly increase the time performance of our implementation.

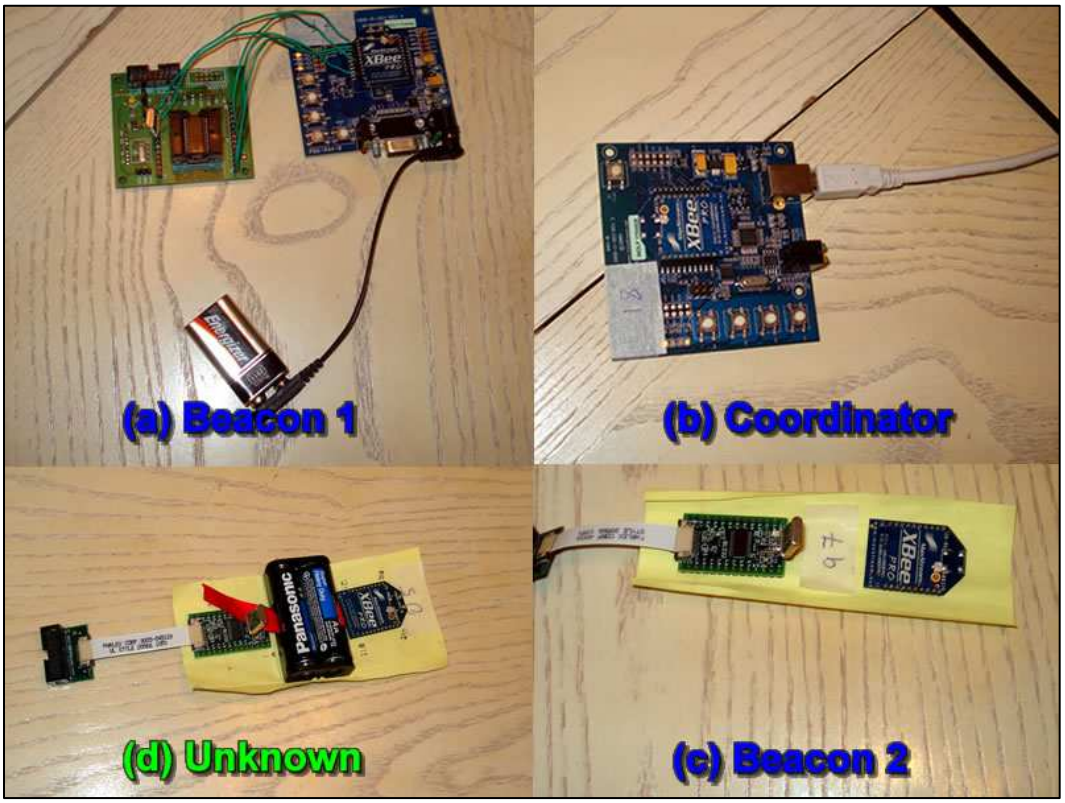

Figure 4.14 Wireless Nodes. (a) Beacon 1, (b) Coordinator, (c) Beacon 2, d) Unknown

Figure 4.14 provides a closer look to our prototype setting. The coordinator node, Figure 4.14.b, is attached to a USB-to-Serial module to interface with the central server PC. Beacon1 is one of the fixed position beacons together with Beacon2. Beacon1 is battery

operated, while Beacon2 is powered from the PC. The last node is Unknown, which is battery operated. This Unknown is the node to be located at the central server. Beacon1, Beacon2 and Unknown are standalone wireless nodes and can operate with a 3.3V power source. They all share the same firmware, configurations and circuit interconnections as described at section 3.1.3.

4.5 Simulation Environment

In order to experiment with networks with larger numbers of nodes we built a simulator environment is shown in Figure 4.15. It is similar to our prototype GUI shown in figure 4.13. There are various elements that are displayed in the map. We start at the center with the beacon antennas shown in blue. These are the three nodes with a known fixed location and one of them starts the broadcasts, just as in a real deployment. The rest of the network is filled with pseudo randomly positioned unknown nodes, shown in green. These nodes have a known location, but the application simulates wireless communications among them and RSS sensing. With this information their location is estimated using the same trilateration and filtration algorithms described in chapter 4. This simulator also shares similar class diagrams as our real prototype. Recently calculated positions are displayed in black and a red line linking the dot to its real position. This red line represents the error in the calculation. As you might observe, nodes far from the center appear to have greater position errors.

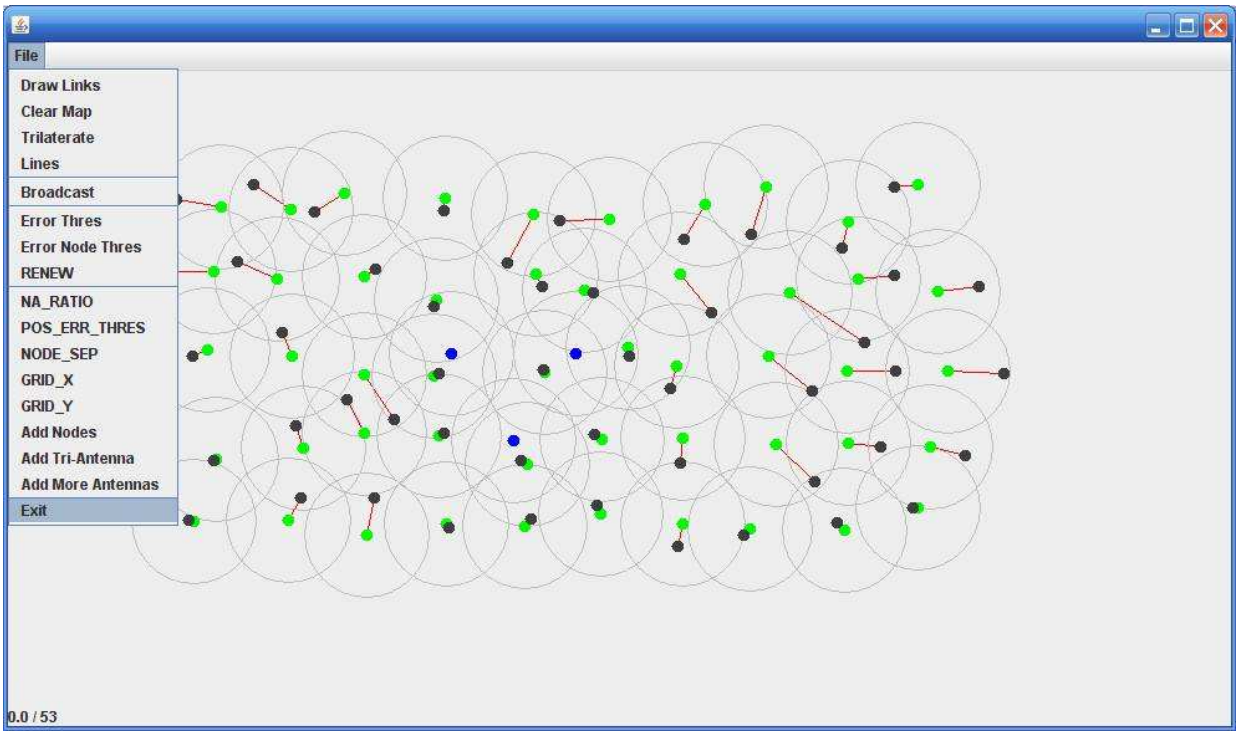

Figure 4.15 Simulation Environment

Several menu options allow the user to modify internal simulation parameters. Some parameters include network size, amount of beacon nodes, positioning error thresholds, node separation and RSS measurements error thresholds. Individual errors and positions are randomized based on the thresholds specified by the user. The user can also insert additional nodes or antennas by clicking with the mouse at a desired position. One last element displayed is the gray circles surrounding each node which represent half their signal radius. The signal radius is internally defined as 100 feet per node.

5 EXPERIMENTAL ANALYSIS

5.1 Introduction

This chapter presents results of experiments conducted on both a working prototype as well as a simulated implementation of our real time location system over wireless mesh networks. The experiments help validate our solutions and ideas, and serve as a demonstration of the feasibility of the system. The idea is to set up a working network of wireless nodes and a central server using our software, and prove it reasonably functional. The specific objectives of the experiments were:

1. To validate the software middleware as a functional system.
2. To validate the prototype deployment location capabilities.
3. To measure the effects of network size on location accuracy.
4. To measure the prototype location latency time.

To test the first, second, and fourth objective, a fixed number of nodes were built to try and locate the position of just one of its wireless nodes. The same location of this node was calculated several times by the central server using different sets of data. Additionally, the location was changed several times the by the central server. Each location was calculated several times using different sets of data. Time was measured at the server for incoming and departing data frames. This experiment is detailed on sections ahead.

The third experiment objective was tested using the simulator developed to test the algorithms. The simulator accepts several parameters such as the network size, error

thresholds, node positioning and number of beacons. We can, in a simulated environment, easily observe the effects of increasing the network size on the position error of a distributed location scheme.

Our central server prototype was developed using Java 1.5 programming language, GNU/RxTx 2.0 Serial COMM interface for Java. The wireless nodes were composed of one DIGI's XBee-PRO wireless module and one TI's MSP430F1232 microcontroller. The XBee modules were using XBEE PRO ZIGBEE COORDINATOR API V.8117 and XBEE PRO ZIGBEE ROUTER API V.8317 firmware code. The MSP430F1232 firmware was developed using The C programming language and compiled with msp-gcc, a GCC toolchain for MSP430. Our network simulator was developed using the Java 1.5 programming language. The experimental environment was composed by:

- One generic PC running Microsoft Windows XP, on an Intel Centrino DUO processor at 1.6 GHz, and 1GB of memory.
- Four XBee-PRO modules each attached to a MSP430F1232 microcontroller running at 4MHz.

5.2 Prototype Implementation Analysis

The objective of this experiment is to verify the prototype deployment location capabilities and at the same time test the software middleware as a functional system. We want to verify that our implementation performs reasonably as expected.

We used the same setup depicted in Figure 4.13 which consisted of three wireless beacons arranged in an equilateral triangle. An additional wireless node was used as the

target for location. Figure 5.1 illustrates the positions and distances of all nodes in this setting. Notice that the distances are in inches, yet this is not the typical real scenario. Remember that our nodes do not have an attached antenna and are working at its lowest power level. Fixed beacons are located twelve inches apart and there are seven important areas of location. The wireless unknown node was positioned at each of these areas and several measurements were taken. The position was manually measured at the setting and manually located at the computer screen, so this might have added some minimal errors to the results.

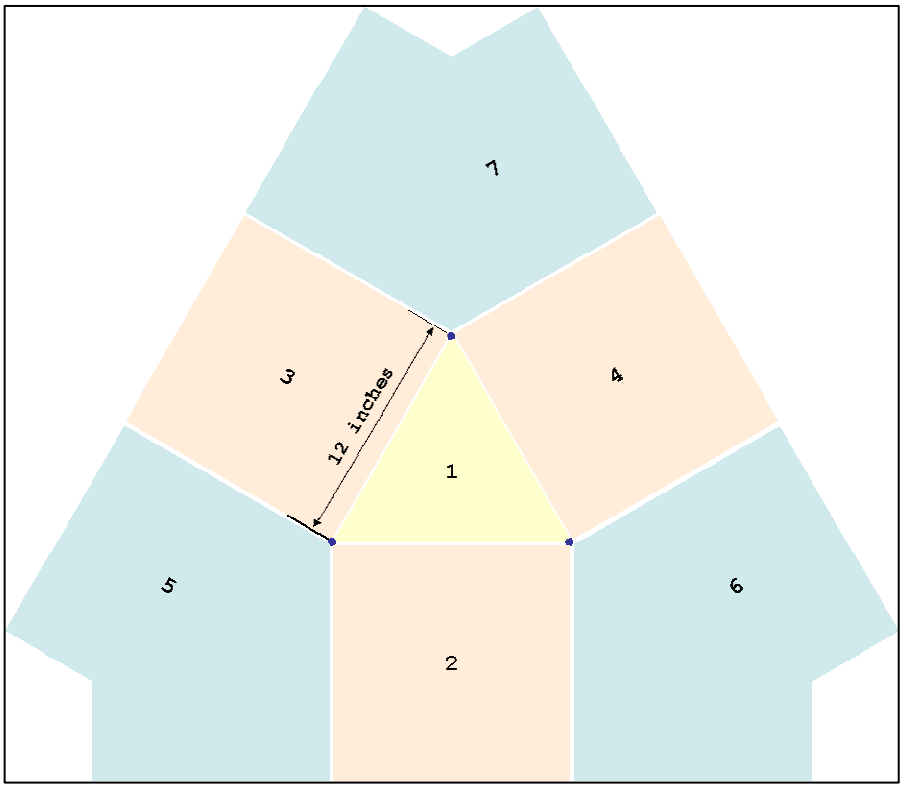

Figure 5.1 Experimental antennas setting

5.2.1 Prototype Location

As stated previously we chose seven areas of location around the beacon antennas. A wireless node was located at each one and about 20 measurements were taken at each location. The measurements are not shown since the data gathered was used internally by the server to locate the nodes. What was finally used as a metric was the (x,y) location of the nodes at the screen of the GUI. The positions were then compared to its approximate exact location and then a distance difference was calculated. This difference was divided by the radius of signal coverage as described in section 4.2 which is about 40 inches.

$$\texttt{distance error} = {}^{(realpos - calcpos)}\!/_{radius} \qquad \textbf{5.1}$$

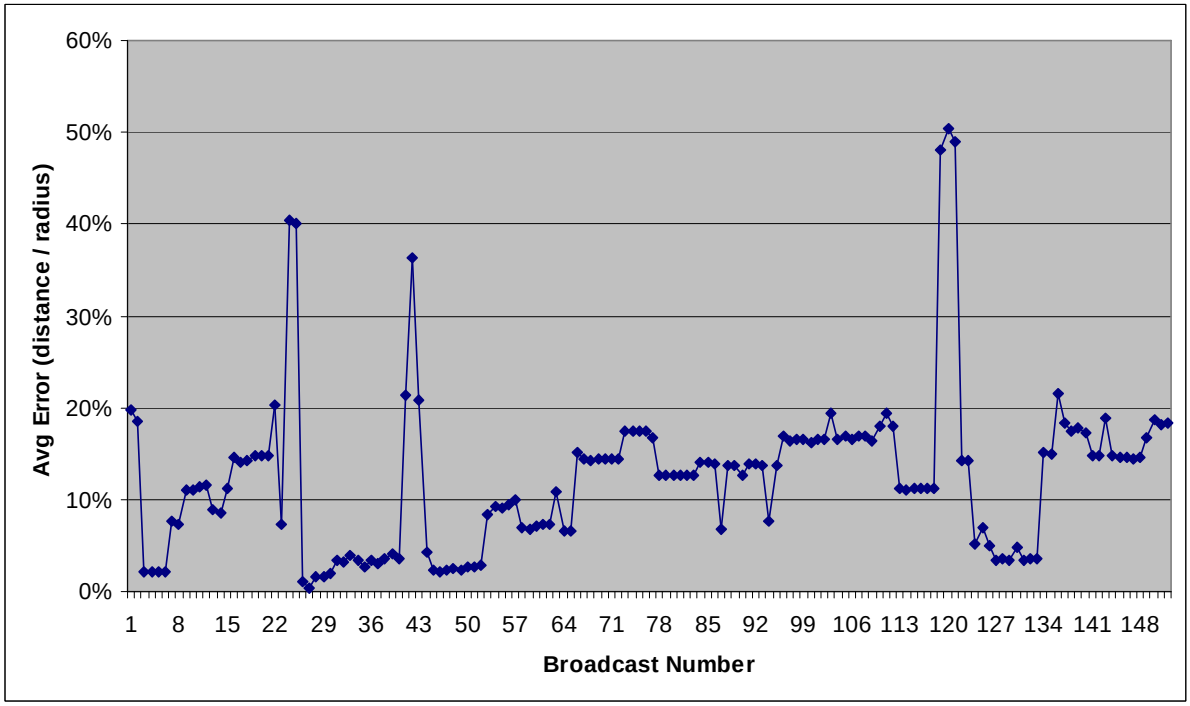

Figure 5.2 Position error at different broadcasts measurements

Figure 5.2 shows the error calculated for each individual measurement as described above. It does not show the position being measured, but it is not important for the analysis. The graph shows an average for all measurements of about 12.42%.

In real world terms, if we had a radius of about 40 inches, then we had about 5 inches of error. Extrapolating to a full blown deployment, the radius is about 100 feet indoors and 12.42 feet of error. Depending on the application this might be acceptable or not, but for many applications we b it is acceptable. Yet we demonstrated the capabilities of our prototype of locating the node at an acceptable margin of about 12%. What was definitive was the functionality of our middleware system. It proved capable of performing as a complete system for wireless node location.

5.3 Effects of Network Size in the Location Error

As shown by our first experiment, our system was capable of handling the data and locating one node within acceptable margins of error. The objective of this second experiment is to determine how position errors affect the localization of other nodes through the network when we increase the network size. This experiment was carried out by using the simulation environment described in section 4.5.

5.3.1 Distributed Location Error Propagation

This second experiment is used to determine how network size affects the average position error obtained. We used the simulator described at section 4.5 for this experiment.

An internal programming interface was used to achieve the same simulation environment without the graphical overload.

In our experiment we defined the parameters as follows.

```
// Nodes to Antennas Ratio
naRatio = 1;
// Position Error Threshold %
posErrThres = 40;
// Extra Antenna separation
nodeSep = 50;
// Tri-antennas?
tri = true;
// More-antennas?
more = false;
// Repeat Broadcast
repsB = 10;
// Iterate Calculations
repsT = 10;
// Error Threshold Sensed RSS 12%
errThresNode = 12;
```

Then we iterated these parameters with several network sizes and obtained the results depicted in Figure 5.3. The ‘Y’ axis shows the average location error measured as a percent of the node signal radius. This means that if the nodes have a signal radius of 100 feet (or 200 feet in diameter), an error of 20% translates to an error of 20 feet. The ‘X’ axis shows the network size in nodes. The network size went from a total of 4 nodes to 579 nodes.

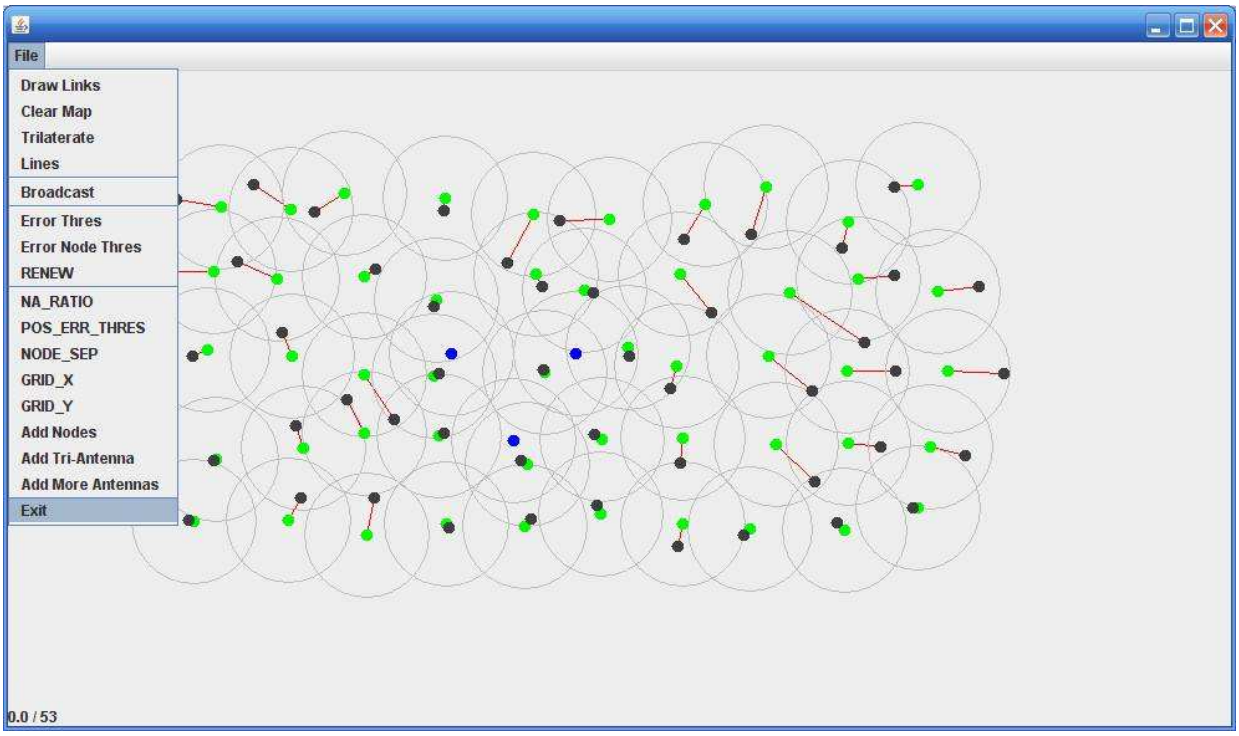

Figure 5.3 Simulation Environment

Notice that our results show an average error of about 20% independent of the network size. If we calculate a global average we obtain 18.69% position error. In practical terms we can conclude that based on these findings, an increase in nodes will not necessarily result in an increase on the average position error.

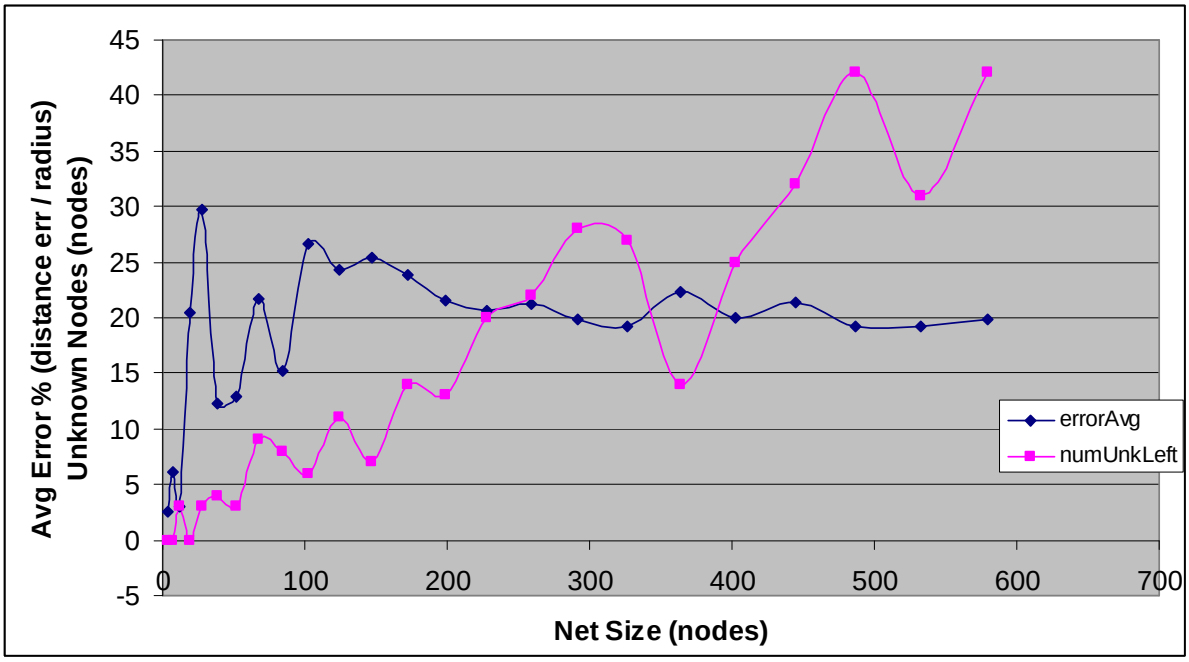

Figure 5.4 Average position error by network size

The graph also shows another line which displays the amount of nodes that are not possible to locate. Notice that the total nodes located is calculated by subtracting this value to the net size. We can observe that it linearly increments with the size of the network, this is mostly the nodes at the edges of the network. The network in this experiment maintains a rectangular shape. As the network size grows the ratio between total nodes and nodes on the edges decreases. This means that there are more nodes inside the network than at its edges as it grows. This appears to be the reason why the average error tends towards a constant value. Future experimentation can verify this claim.

5.4 Location Latency

The objective of this third experiment is to measure the location latency time of our experimental prototype. This is the time it takes the system to refresh the location of a moving node. This value could be used to determine the minimum time rate at which the server should send location broadcasts.

We used the same setup depicted in Figure 4.13 which consisted of three wireless beacons arranged in an equilateral triangle. An additional wireless node was used as the subject for location. We measured latency by measuring the time it took a broadcast frame to propagate trough the network and the time the central server received responses from all nodes in the network. The experiment consisted of 18 consecutive location broadcasts. Figure 5.5 shows our results.

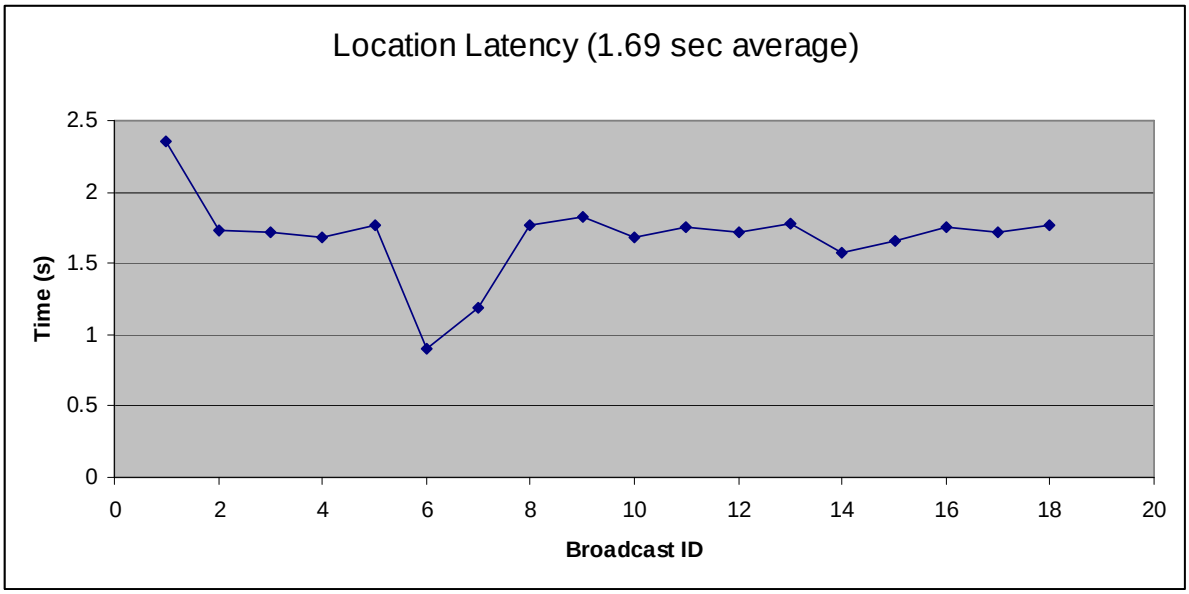

Figure 5.5 Location Latency

We found out that the prototype is capable of refreshing the node's position at an average of 1.69 seconds. The ideal latency value for a Real Time Location System is zero. As implemented, this latency will increase as the number of nodes increases in the network. Nodes deeper in the network will have greater latency that those close to the central server antenna.

Latency can be improved by increasing the network baud rate, by increasing the central server's antenna coverage, or by installing additional antennas directly connected to the central server and scattered deeper in the network.

5.5 Cost and Power Analysis

In this section we present a rough BOM (Bill of Materials) cost and an approximation of power consumption for the wireless nodes. An approximate range of prices is given as they were found at online catalogs. The prices are current. A wireless node consists of an XBee module, an MSP430 microcontroller, an 8MHz crystal, a PCB board, a 500 mAh coin battery, a battery holder and a plastic enclosure. The PCB board and the enclosure can be custom made but the prices presented are off-the-shelf products. Each node ranges from $25.28 – $40.51 total cost in materials.

XBee Module	$19 – $32	x 1
MSP430 MCU	$1.05 – $1.87	x 1
8MHz Cristal	$0.22 – $0.32	x 1
PCB Board	$4.19 – $4.50	/ 20
Battery (500 mAh)	$2.56 – $3.20	x 1 (10 hr @ full power)
Battery Holder	$0.75 – $1.10	x 1
Enclosure	$1.49 – $1.79	x 1
Wireless Node	**$25.28 – $40.51**	

The best way to estimate the power consumption is to experimentally measure the current consumption of the device under different scenarios. We roughly estimate the power consumption based on manufacturer specifications. The XBee module consumes about 50mA when transmitting and receiving and the MSP430 consumes about 200 uA at full power. Based on these numbers we compute their power consumption as 50.2 mA when a data frame is received or transmitted. The amount of power consumed at sleep mode is negligible. To calculate the time spent receiving or transmitting we calculate the time needed to process frames of 160 – 176 bits long at 9600 baud rate. We get a total of 35 ms if a frame is received and resent.

Transmit Rx	22 bytes	176 bits	18.3 ms @ 9600 baud
Receive Pckt	20 bytes	160 bits	16.7 ms @ 9600 baud
Full Power			**35 ms**

At full power the node consumes 50.2 mA and drains a coin battery with a capacity of 500 mAh in about 10 hours. But this is not the case since most of the time the nodes are at sleep mode. To get realistic numbers we use the time spent transmitting and receiving which is about 35ms. We will call this transmit/receive cycle a Tx/Rx slot. Given a total of 10 hours of battery power, we get 1028571 Tx/Rx slots.

10 hours x 60 min x 60 sec x 100 ms / 35 ms = 1028571 Tx/Rx slots

For each server broadcast we can assume that each node spends 3 Tx/Rx slots receiving the broadcast and sending RSS data back to the server. We get 342857 Tx/Rx broadcast slots (TRBS). This means that each node lasts for 342,857 server broadcasts after the battery is drained.

1028571 Tx/Rx slots / 3 per broadcast = 342857 Tx/Rx broadcast slots (TRBS)

If we set a 10 second time interval between each server broadcast, we get 952 hours of operation for a node which translates to about 40 days of operation with a coin battery.

342857 (TRBS) x 10 second broadcast interval = 952 hours ~ 40 days

These figures can improve by increasing the network baud rate and decreasing the broadcast interval. The figures relate to the prototype implemented. We do not claim the correctness of these calculations. They are intended to be used as a guide.

6 CONCLUSIONS

In this dissertation we have presented a real time location system built on top of a wireless mesh network. This system builds on top of the basic infrastructure for RTLS solutions, including wireless nodes, wireless beacon nodes, a central server and a programming interface for the XBee module. We described the motivations behind building a new RTLS solution, its numerous applications as well as the potential for future markets. The system architecture and mathematical procedures were described in some detail. We presented a working prototype implementation of the system, a software simulator and conducted experiments that demonstrated its functionality and performance.

6.1 Summary of Contributions

The work presented describes and demonstrates our solution to real time location. We have been able to produce a software middleware and infrastructure for future RTLS applications and developments. Our system could be applied and deployed to numerous kinds of applications and problems requiring low-cost, small foot print and easily maintainable and scaleable infrastructure.

In Chapter 3, we presented the design of our real time location system infrastructure. We started by presenting the hardware components and technologies used on our design and

the reasons we selected them. First we presented the selected transceivers, microcontrollers, and the network protocol.

- *XBee Wireless Module* – It is a stand-alone, ready-to-use solution for low-power, low-cost, wireless sensor networks. They operate within the ZigBee mesh networking protocol at the network level or MAC level.
- *MSP430* – The MSP430 is an ultra low-power microcontroller and consists of several devices featuring different sets of peripherals targeted for various applications.

Then we discussed some issues concerning the distance measurement method.

- *Received Signal Strength* (RSS) – RSS can be used as a metric of distance from node to node. The loss on the signal can be translated to an approximate of the distance between the measuring nodes.

Finally we discussed the centralized real time location scheme.

- *Cooperative Multilateration* – Using cooperative multilateration most nodes in the network could be located within an average error of 12% of the coverage radio, which is acceptable for many applications. Cooperative multilateration works by locating the unknown nodes closer to at least three beacons. This is accomplished by trigonometric multilateration using the distance among nodes as measured by the received signal strength. Their location is then propagated to the farther unknown nodes by using the location of the newly located nodes.

In Chapter 4 we described our system design. We described the wireless nodes behavior and internal algorithms.

- *Wireless Nodes* – The nodes have several tasks to do besides creating and maintaining the wireless mesh network. Each node receives and performs commands sent by the central server, also return RSS values to the central server and propagate server broadcasts.
- *Central Server* – The central server software is in charge of managing the coordinator node, configuring the wireless nodes in the network, and calculating the location of all nodes. Location information can be used to draw the location of wireless nodes in a map.

We then described the central server data gathering, trilateration and filtration algorithms. We also described the data flow in the network of nodes. Finally we explained the relationship between RSS and distance among nodes.

In Chapter 5 we presented a working implementation of our proposed software middleware for real time location systems on wireless mesh networks. The experiments helped validate our solutions and ideas and served as a demonstration of the feasibility of building such a system. The experiments helped us validate the software middleware as a functional system, the prototype deployment location capabilities, and to measure the effects of network size in the location error. We concluded that based on the findings, an increase in nodes will not necessarily result in an increase on the average position error. We also demonstrated the capabilities of our prototype to locate nodes at a margin of about 12% of the coverage radio. Prototype tests also showed that the system had an average location

latency of 1.69 seconds. Our work demonstrated the feasibility of RTL with of-the-shelf components. A brief analysis on costs showed that wireless nodes could be manufactured at around $25. A theoretical analysis on power consumption showed that a node could drain a coin sized battery in about 40 days if the system refreshed locations at 10 seconds intervals.

6.2 Future Work

The following is a partial list of interesting future developments for our work:

- *Wireless Nodes* – Wireless nodes firmware could be greatly improved on several areas regarding power management, communication speed improvements, RSS sensing precision, calculation speed improvements, parallel RSS sensing and return, frame loss improvements, extension on XBee module management, and the ability of remote I/O manipulation. This last one is an interesting and useful topic since it enables a remote manipulation of the microcontroller's internal modules. It adds a variety of capabilities to the system besides location.
- *Wireless Nodes Packaging*– A prototype package with minimal size, antenna protection, coin battery, weather protection, and external ports.
- *Central Server* – The central server should implement a generic communication interface to enable multilanguage programming. Improvements should be made to data filters and calculation speeds. A three dimensional trilateration scheme is greatly suggested. Also multiple points of RSS data collection are a major improvement to data collection.

- *Moving Beacons* – Explore the behavior of the system by introducing some nodes equipped with a GPS device and the capability to move over the map.

REFERENCES

[1] N. Patwari, A.O. Hero III, J. Ash, R.L. Moses, S. Kyperountas, and N.S. Correal, "Locating the nodes,'' IEEE Signal Processing Mag., vol. 22, no. 4, pp. 54–69, July 2005.

[2] K Savvides, W. Garber, S. Adlaha R. Moses, and M. Srivastava. "On the Error Characteristics of Multihop Node Localization in Wireless Sensor Networks", Proceedings of First International Workshop on Information Processing in Sensor Networks, 2003.

[3] P. Harrop and Raghu Das. "Active RFID and Sensor Networks 2008-2018: Rapidly growing sector", IDTechEx Ltd, Cambridge, MA, 2008

[4] Gezici, S.; Zhi Tian; Giannakis, G.B.; Kobayashi, H.; Molisch, A.F.; Poor, H.V.; Sahinoglu, Z., "Localization via ultra-wideband radios: a look at positioning aspects for future sensor networks," *Signal Processing Magazine, IEEE,* On page(s): 70- 84, Volume: 22, Issue: 4, July 2005

[5] Xiaoli Li , Hongchi Shi , Yi Shang, "A Sorted RSS Quantization Based Algorithm for Sensor Network Localization", Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS'05), p.557-563, July 20-22, 2005

[6] Kliment Yanev, "Location-aware computing", University of Helsinki, Department of Computer Science

[7] Y. Gwon, R. Jain, and T. Kawahara. "Robust indoor location estimation of stationary and mobile users", In Proceedings The 23rd Conference of the IEEE Communications Society (INFOCOM), Hong Kong, Mar. 2004.

[8] A. Savvides, H. Park, M. Srivastava, "The bits and flops of the N-hop multilateration primitive for node localization problems", in: First ACM International Workshop on Wireless Sensor Networks and Application (WSNA), Atlanta, GA, 2002, pp. 112–121

[9] Andreas Savvides , Chih-Chieh Han , Mani B. Strivastava, "Dynamic fine-grained localization in Ad-Hoc networks of sensors", Proceedings of the 7th annual international conference on Mobile computing and networking, p.166-179, July 2001, Rome, Italy

[10] Challa, S.; Leipold, F.; Deshpande, S.K.; Liu, M., "Simultaneous Localization and Mapping in Wireless Sensor Networks", Intelligent Sensors, Sensor Networks and Information Processing Conference, 2005. Proceedings of the 2005 International Conference on, Vol., Iss., 5-8 Dec. 2005 Pages: 81- 87

[11] N. Moore, Y. A. S¸ekercio˘glu, and G. K. Egan, “Virtual localization for mesh network routing”, Proceedings of IASTED International Conference on Networks and Communication Systems (NCS2005), April 2005.

[12] Weidong Wang; Qingxin Zhu, “High Accuracy Geometric Localization Scheme for Wireless Sensor Networks”, Communications, Circuits and Systems Proceedings, 2006 International Conference on, Vol.3, Iss., June 2006 Pages:1507-1512

[13] M. Sichitiu, V. Ramadurai, “Localization of Wireless Sensor Networks with a Mobile Beacon”, in Proceedings of MASS, 2004.

[14] Ouri Wolfson, “Moving Objects Information Management: The Database Challenge”, Department of Computer Science, University of Illinois, Chicago, IL.

[15] J. Hightower, G. Borriello, “Location Systems for Ubiquitous Computing”, Washington University, WA, 2001.

[16] ZigBee Alliance, “ZigBee Specification Version 1.0.”, San Ramon, CA, USA, December. 2004. [Online]. Available: http://www.zigbee.org/en/spec_download