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
_____	_____
Wanda Negrón, PhD Representative of Graduate Studies	Date
_____	_____
Isidoro Couvertier-Reyes, PhD Chairperson of the Department	Date

ABSTRACT

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 technology as the tools for design and development. Arguments are made for its benefits on ease of development and deployment costs. Also several tests were conducted on a preliminary experimental prototype. These tests demonstrate the capabilities and functionality of the system.

RESUMEN

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 trough 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 to 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 cherish.

To all of you, I dedicate this work.

Table of Contents

Figure List

1 INTRODUCTION

Node location in wireless mesh networks is an indispensable requirement for a whole array of applications. Node location has uses in network routing optimization, asset tracking, asset identification and other areas. As low-cost and low power sensor networks become main-stream, an accurate, low-power and low cost scheme for sensor location is needed. This kind of peer-to-peer, or mesh, networks provide an inherent flexibility in terms of network topology. The location technique and algorithm should be modeled based on the requirement of the application such as accuracy, energy efficiency, scalability, or cost. There are various distance measurement techniques that could be used for wireless mesh networks location, being the most practical for our particular interest 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 analysis.

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 take 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 computers nodes that interact with each other to share gathered information, execute commands and most importantly to communicate. In the vast ocean of applications and

purpose of wireless sensor networks, this paper will focus on how sensor networks determine their physical location in the network.

1.1.1 Applications

Here we present a minimal subset of applications and possibilities for wireless mesh network node location. Their importance will become obvious as they get mentioned.

1.1.1.1 Animal Tracking

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 in different scenarios.

1.1.1.2 Personnel Tracking

As stated before, people location can be critical for the operations of business and events. A hospital is a good example where knowing who and where is the closest doctor or nurse can be decisive 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 greatly benefit event's goals.

1.1.1.4 Industrial / Home Automation

Ubiquitous computing is one of the most hyped areas lately. 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 other likes for numerous of reasons. Take for example the space exploration. A swarm of robots could be deployed in a vast area. All their interaction with their environment and themselves must include the awareness of location to maximize their collaboration and scarce resources use.

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. In [3] 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 relates to Real Time Locating Systems (RTLS), Ubiquitous Sensor Networks and active RFID tags based on Zigbee, 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.

Currently, the RTLS market consists on whole systems rather than generic software applications for generic hardware infrastructures.

1.2 Proposed Solution

We want to build a basic, low cost infrastructure for RTLS based on wireless mesh networks. The specifics of the network can be modeled based on the applications, and the applications are influenced on the market. The hardware infrastructure must be built upon off-the-shelf solutions. 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 hardware infrastructure. RTLS can aid in cattle theft prevention, family monitoring in an amusement park, personnel location in a theatrical or musical event, soldier assistance in the battle ground, and even in the orchestration of a swarm of house keeping robots.

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

1.3 Objectives of this Thesis

The main objectives of this Thesis are:

- To design a real time location system based on a wireless mesh network and using the received signal strength as the method of distance measurement.

- To develop an implementation of the real time location system using off-the-shelf components.
- To develop an extensible software middleware for real time location system and related applications.
- To investigate the reliability of RTLS based on wireless mesh networks versus other approaches to the problem.

1.4 Contributions

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

- A solution to real time location system based on wireless mesh network is presented using off-the-shelf technology as the tools for design and development. Arguments are made for its benefits on ease of development and deployment costs.
- An implementation of a software middleware for RTLS is presented as well the hardware infrastructure for the underlying wireless network.
- An implementation of a theoretical environment for simulations was developed to test methodology and algorithms.
- Several tests were conducted on a preliminary experimental prototype. These tests demonstrate the capabilities and functionality of the system.

- Several tests were conducted on a theoretical environment to prove the correctness of our approach and algorithms.

1.5 Thesis Structure

The remainder of this thesis is structured as follows. In Chapter two we present a survey of the most relevant technologies related to our research, 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 presents the results from several experiments related to the performance of the system. Finally chapter six present a summary of contributions, conclusions and directions regarding future work.

2 RELATED WORK

2.1 Measurements 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 circuitry, the received signal strength (RSS), time of arrival (TOA), difference on time of arrival (TDOA), and angle of arrival (AOA) information can be used, or a combination of some of them. Following is the description of the previous.

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 translated to an approximate of the distance between the measuring nodes. It is widely used as a measurement technique since the hardware that implements it consists on very simple and inexpensive circuitry [5]. It is available on most commercial transceivers, thus its importance. Yet these types of measures suffer from an infamous unpredictability. This is due to many sources of error.

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 plus a propagation time delay. This measured delay between sender and receiver is directly translated to their separation distance divided to the propagation velocity. The signal could be acoustic, RF or any other. 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. This way an asynchronous sensor can be used. A constant delay from the re-sender might be taken in to account.

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. The technique alone cannot determine distance but can dramatically improve TOA or RSS calculations.

2.1.4 Connectivity

Connectivity is a simple binary value that only tells whether another device is on 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 its signal range and makes a relation among 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

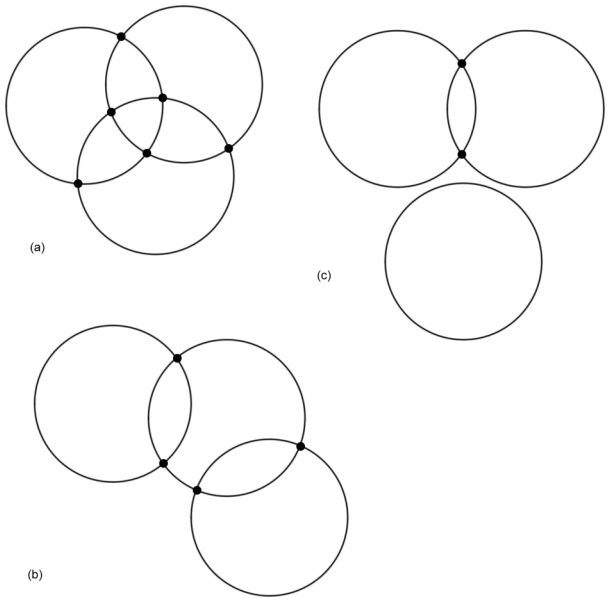

Figure 2.1 Common circle patterns. (a) Type 1 (b) Type 2 (c) Type 3

Triangulation is a method that solves a set of linear equations involving the coordinates of multiple reference points (Xi, Yi) at known coordinates and measurements of distance Di to these points [6]. A node can estimate its position given three or more reference

points in range. It is limited by the precision of distance measurements and the reference point accuracy. The measurements would be based on RSS, TOA, TDOA or AOA.

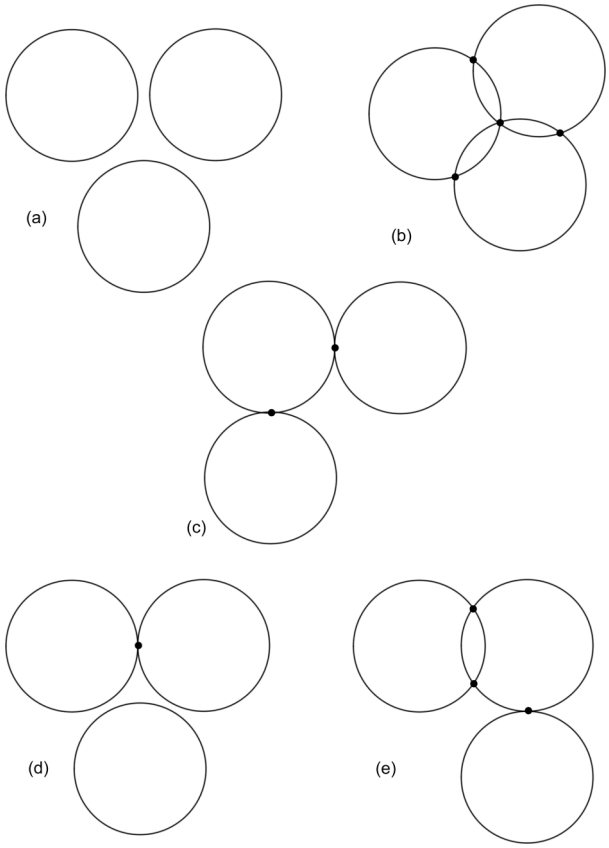

Figure 2.2 Uncommon circle patterns

Triangulation can result in various patterns of circles. The patterns can be classified depending on the number of real and imaginary solutions to the system of equations [7]. The most common pattern of circles are shown in Figure 2.1. As shown by [7], Type 1 has six real solutions, Type 2 has four real solutions and three imaginary solutions, and Type 3 has two real and four imaginary solutions. Uncommon circle pattern are shown in Figure 2.2. Figure 2.2 (a) shows a pattern with no solutions, (b) show the most desirable pattern, the

triple root point, (c) two double roots and two imaginary solutions, (d) one double root and four imaginary solutions, and (e) two real solutions and one double root.

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 obvious 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 Maximum Likelihood

Maximum likelihood works by finding a point that stays the closest to as many as possible of the nodes of a given set of measurements. It chooses a position so that the differences between estimated and measured distances are minimized [8].

2.2.4 Cooperative Location Topologies

Cooperative location is needed when there are 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 being discussed in here take different approaches including a hybrid location topology where some of the calculations are determined locally, hence distributed, and final calculations are determined at a central node. The solution will depend on the specific needs of the system.

2.2.4.1 Centralized

Centralized topologies work by taking the measures at the nodes and then sending all the data 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. There could be other constrains not mentioned here. Otherwise a distributed topology is more appropriate for location algorithms [10, 11].

2.2.4.2 Distributed

Distributed topologies are needed when our specifics include limited communication bandwidth and a lack of 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 the algorithm used, the minimum amount of beacons needed is three to localize the most of the network. All algorithms based on network/cooperative/distributed multilateration suffer from a greater amount of error accumulation than atomic multilateration. Successive refinement could be a method used to minimize the error accumulation impact upon location estimates. Also hybrid methods could improve on this 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. Following are example algorithms that implement what was described before.

2.2.5 Ranging

[12] presents 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 into 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 approximate location.

3 DESIGN

Here 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 important aspects of the technologies.

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

3.1 Off-the-shelf Technology

In the world of silicon and integrated circuits we can find more solutions than the problems we have. This in itself could become a problem. We want to design generic software for specific devices that not necessarily are standardized. 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 to be low-cost, low-power, and mesh-networked. Other requirements were drawn upon several iterations of the design and a specific application as the motivation. The other requirements are to have a centralized location server, to use the signal strength as a measurement of distance, and to use standardized technologies. Based on these requirements

we have selected Digi's XBee wireless module and TI'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 solution for low-power, low-cost, wireless sensor networks. They operate within the ZigBee mesh networking protocol at the network level or MAC level. The network could grow up to 65,000 wireless nodes 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.

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 RSSI 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 RSSI 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 RSSI PWM signal.

3.1.3 Hardware Interface

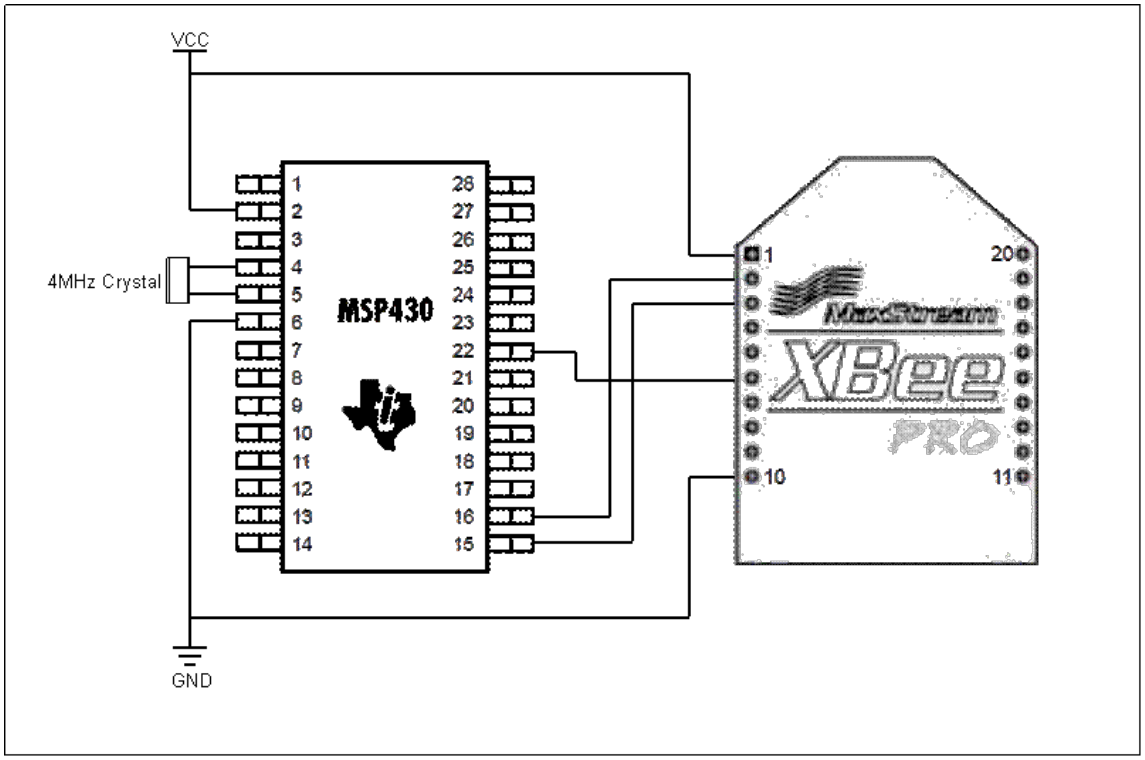

Figure 3.1 Hardware Block Diagram

Figure 3.1 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 RSSI pin. The resulting device might not be much larger than the XBee module including a coin sized battery. Figure 3.2 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.

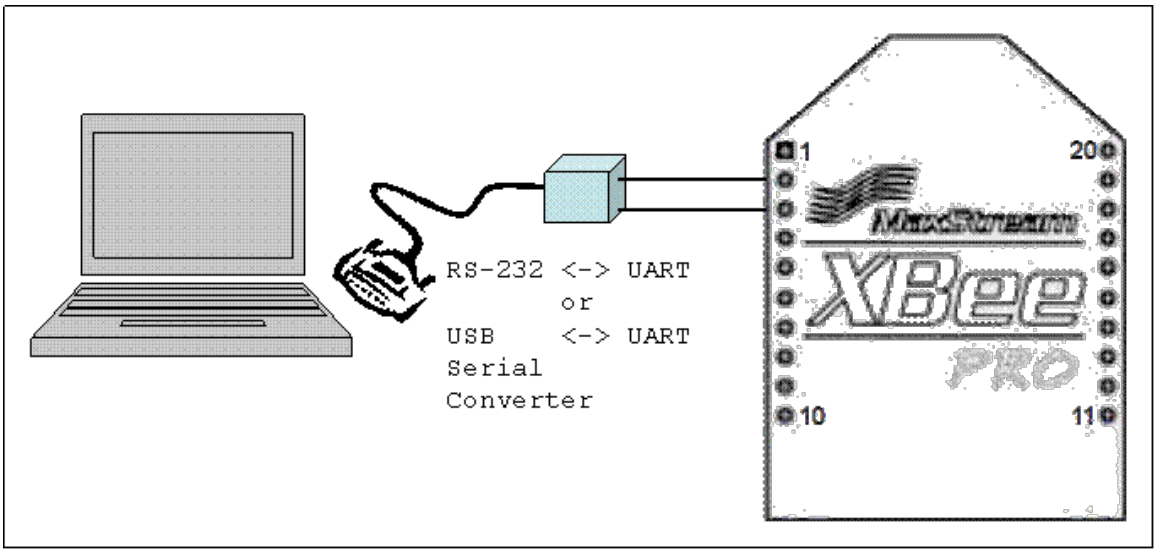

Figure 3.2 PC – Coordinator Interface

3.2 Zigbee Mesh Networking Protocol

The importance of mesh networking relies on the ease of deployment and maintenance of the network. We have selected the ZigBee mesh networking protocol 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 3.3.

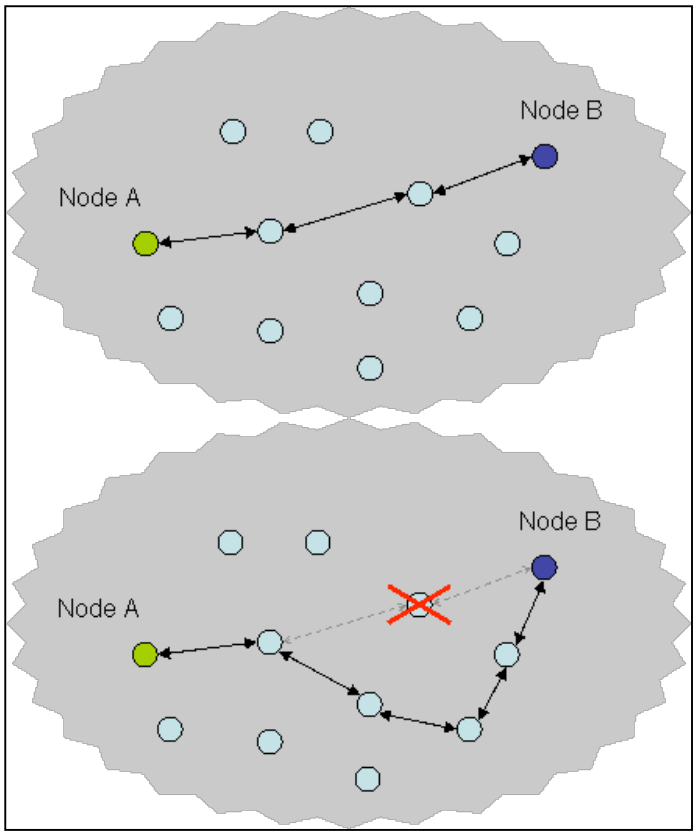

Figure 3.3 Self Healing Mesh Network

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 translated to an approximate 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, namely XBee. The greatest problem with this kind of measurement is that it suffers unpredictability.

This is due to many 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 spread-spectrum method could be used 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 where the signal muss pass trough or diffract around the object. They are considered a random error source. Other sources of error could be attributed to measurement circuitry precision and calibration but are considered insignificant.

RSS errors are considered to be multiplicative, in comparison to other techniques which error sources are considered to be additive. 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. This method yields acceptable results.

3.4 Centralized Location

Mesh networking is by definition a 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 fixed and 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 3.4 shows the way location spreads.

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.

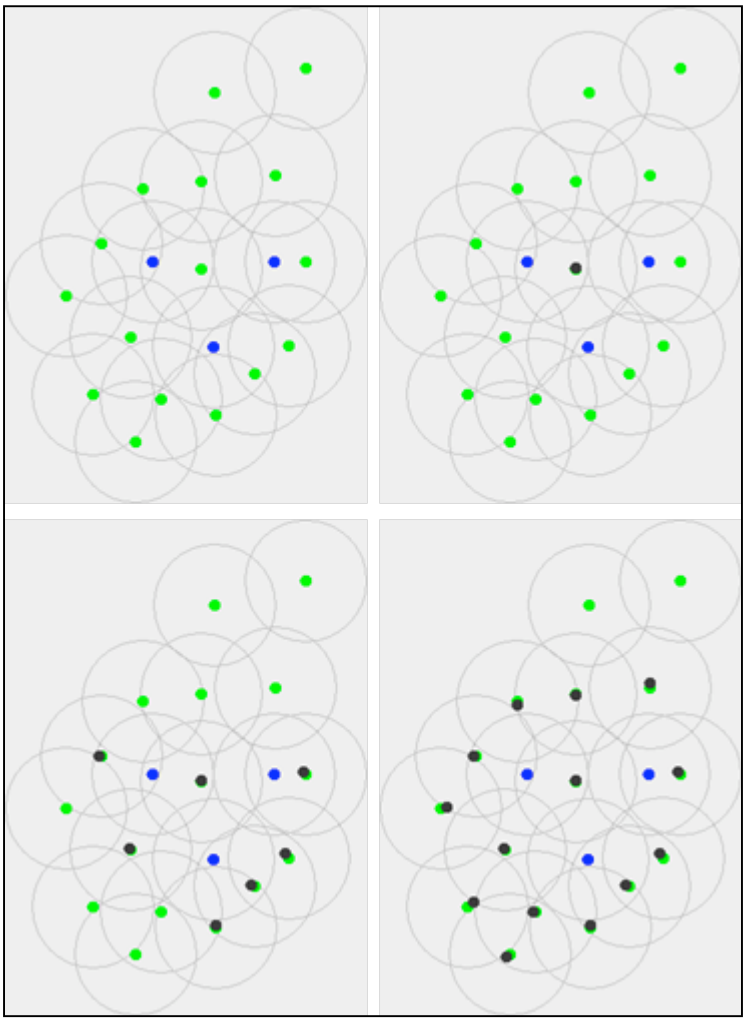

Figure 3.4 Location Propagation

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.

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 centralize 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. 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 connected, and can give useful information.

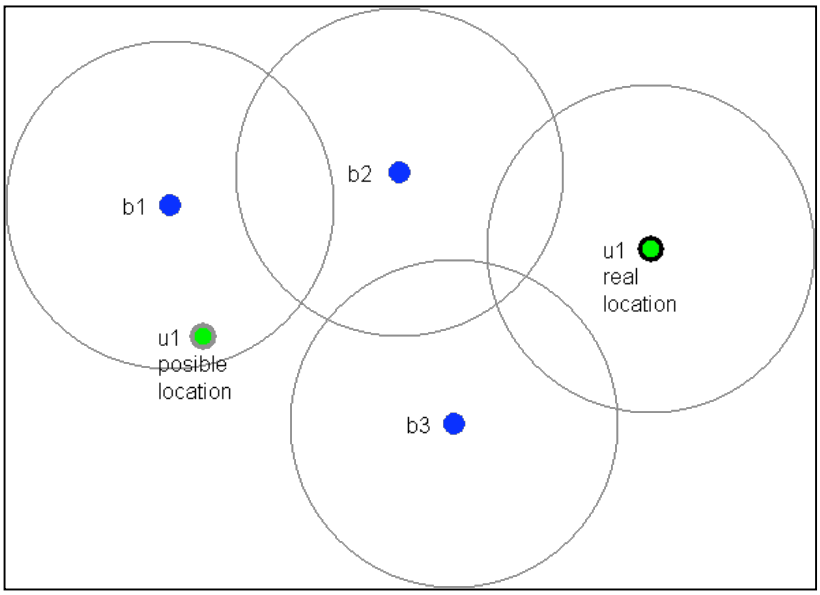

Figure 3.5 Special Case

Figure 3.5 shows a case where we can make a good prediction of an unknown node at a centralized server, but not at the node level. 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.

4 IMPLEMENTATION

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. Following that is the central server data gathering, trilateration and filtration algorithms. We will also describe the data flow in the network of nodes. Finally we will explain the relationship between the RSSI and the distance among nodes.

4.1 Trilateration Algorithm

As stated in chapter 2, trilateration is a geometric method to solve location problems. It could be used to solve two dimensional location as well as three dimensional locations. Both use similar formulas and techniques, but we will use two dimensional data as far as our research concerns. 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.

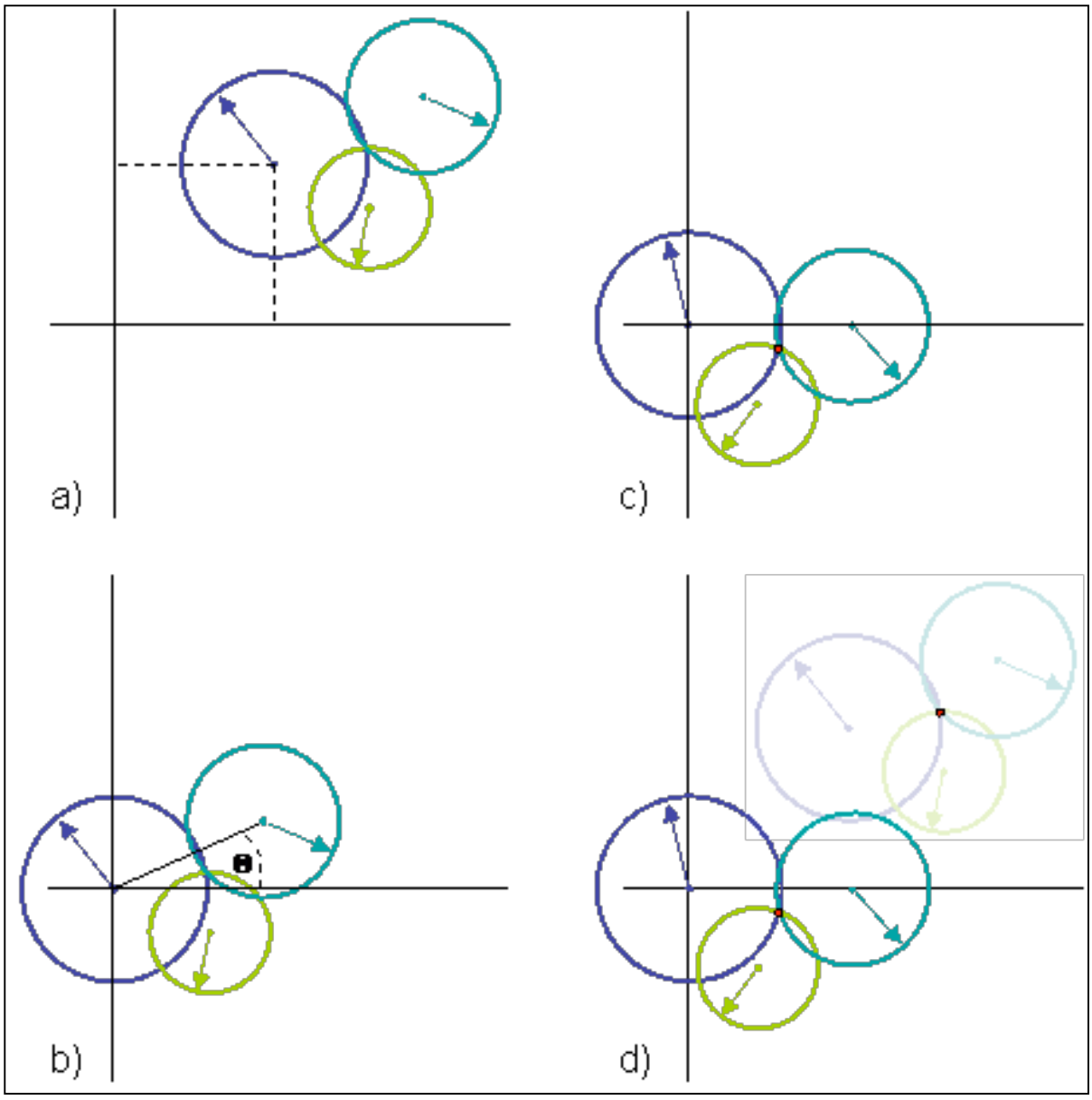

Figure 4.1 Trilateration

Figure 4.1.a shows three circles and a point equal to the three. Our goal is to find the coordinate where all three circles are equal, 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.

```
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 displace `xAxis` circle to the x axis. This is shown in Figure 4.1.c.

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

```
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 equaling 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 `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 RSSI to Distance Relation Characterization

We have already have 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 RSSI 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 neither we are defining one since it is out of the scope of this research. Instead we can characterize a model using experimental data for a given environment.

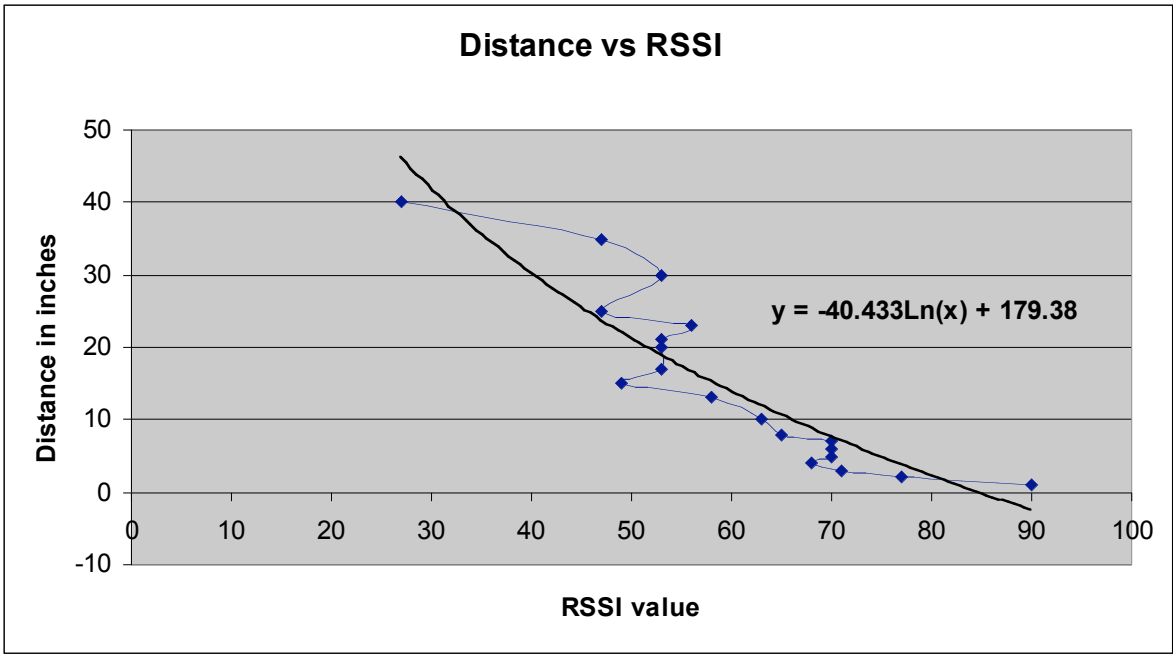

Figure 4.2 Distance vs. RSSI characterization

Figure 4.1 shows the characterization of one our experimental settings. 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 RSSI 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 RSSI 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.

4.3 Wireless Nodes Firmware

This section will briefly explain the software developed to run on the wireless nodes and their behavior. The code has been simplified to clearly understand 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, also return RSSI 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.

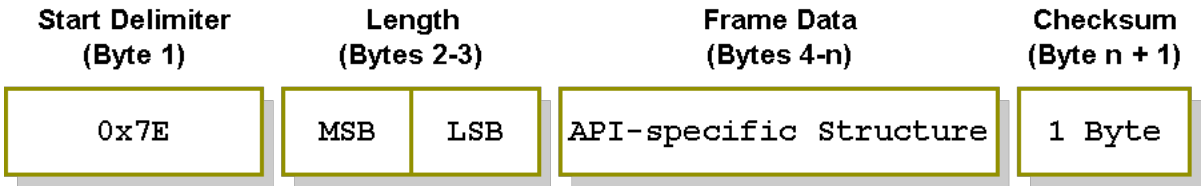

Figure 4.3 General XBee API Frame Structure

4.3.2 Main Algorithm

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

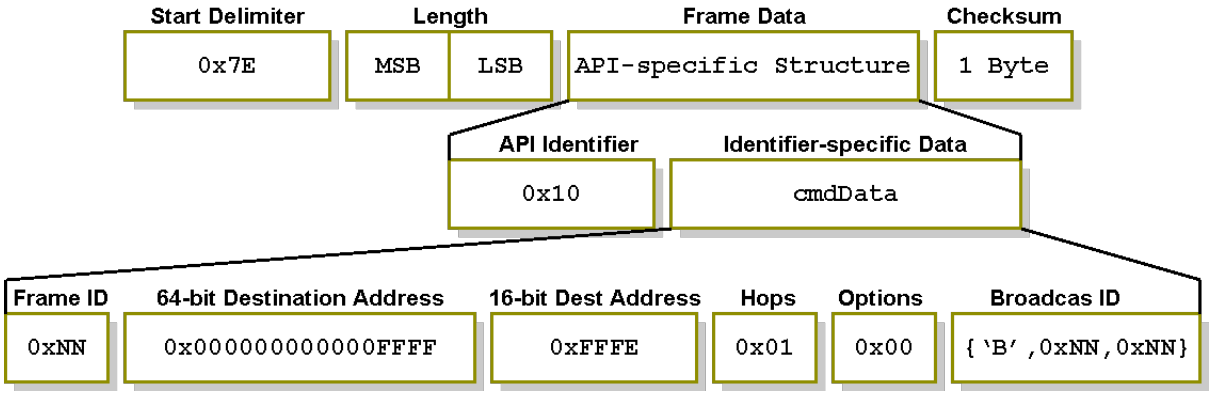

Figure 4.4 Broadcast Frame

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 occurs at the network level. 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. To clear out the algorithms let's describe it step by step.

```
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 RSSI value for this last frame
      senseRSSI();
      // Prepare RSSI return frame
      returnRSSIFrame[];
      // Send RSSI return frame to central server
      sendFrame(returnRSSIFrame);
    }
    // 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 part of the internal algorithm executed by the microcontroller. The pseudo code is shown. We do not have an operating system running on the microcontrollers.

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 at 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 parsed 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 Listing X, execution stops until an event occurs. The microcontroller leaves a 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, main loop continues execution. It starts by verifying if a broadcast was received. If it was received, the RSSI must be determined and sent back to the central server. It will be described in detail in the next section. After this

step, it is determined 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. The last step works with commands. If the frame contained a command, it is executed. Most commands are directed to the XBee module to modify some property of it.

4.3.3 RSSI Sensing

```
senseRSSI(){
  // 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 RSSI value and return
  calculateRSSI();
}

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 RSSI sensing pseudo-code

Figure 4.6 shows the pseudo code for RSSI 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 do some calculations and return. That simple. Now

let's examine the timer settings. As stated previously, the XBee module outputs the RSSI data via a PWM signal.

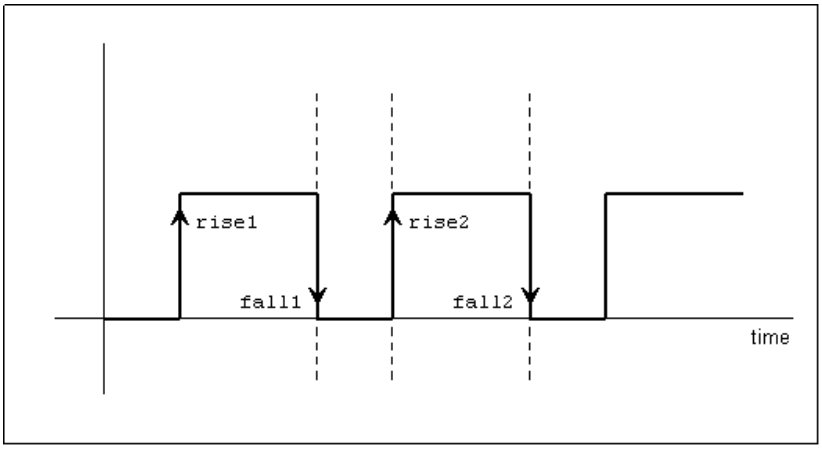

Figure 4.7 PWM RSSI signal

Figure 4.7 shows an example of such a 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 RSSI value.

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. 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. It is not necessary to go into details.

4.4 Central Server Application

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. Our implementation includes a basic two dimensional map. This user interface will be described briefly on sections ahead.

4.4.1 XBee Software Interface

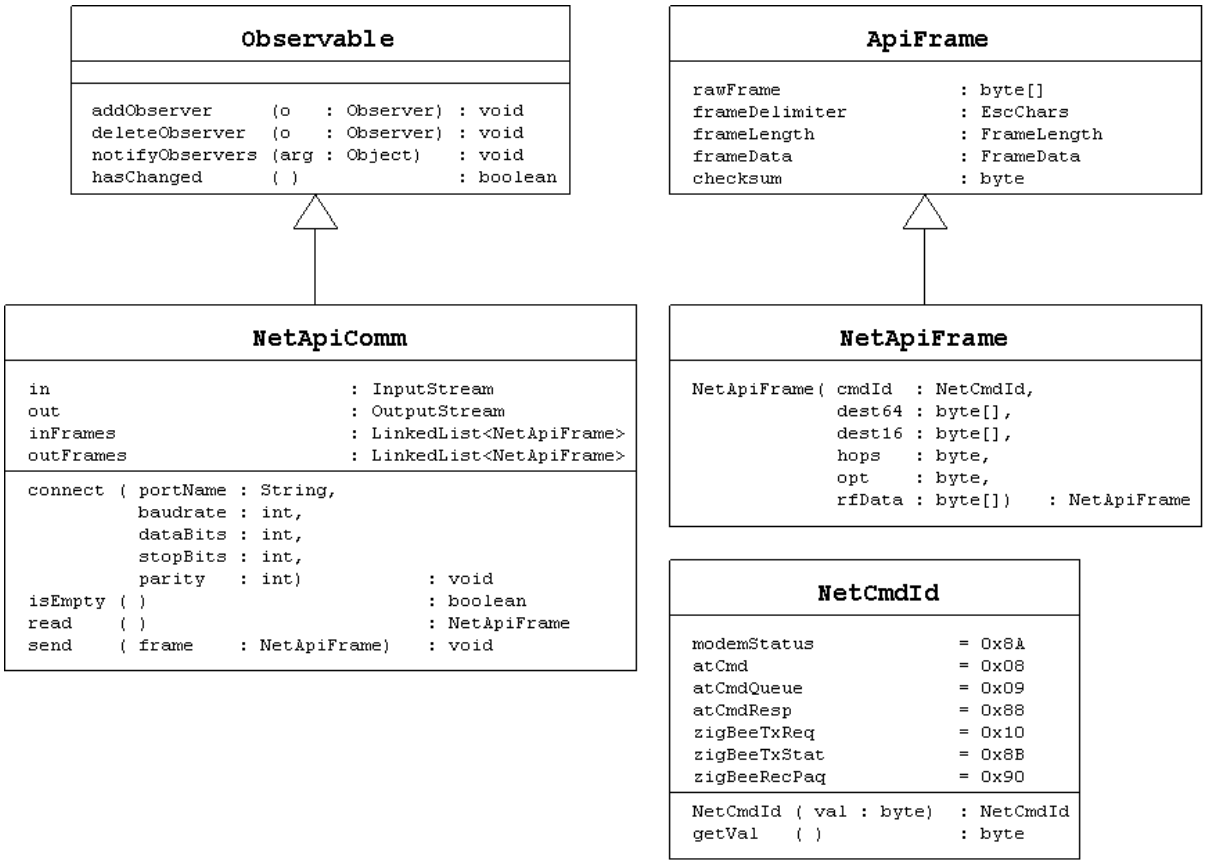

Figure 4.9 Simplified XBee Software Interface Class Diagram

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 `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 `NetApiFrame` class. It defines a general frame structure as shown in Figure 4.3. All frame types and command types have been defined and implemented. We also did 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 RSSI sensing process and the return the information to the central server. The broadcast and return process continues at intervals defined by the server.

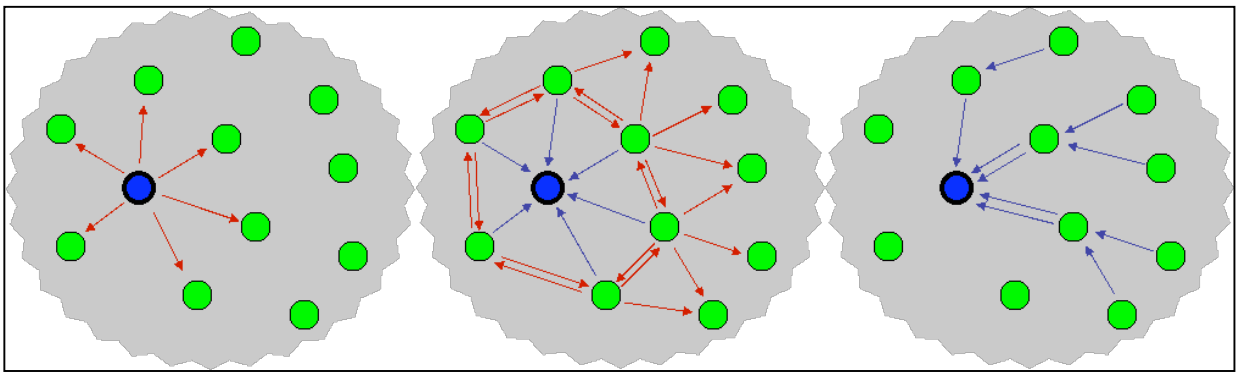

Figure 4.10 Broadcast and Return Data Flow

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 RSSI among beacons to adjust the mapped distance at the human interface map. Red arrows represent the spreading of the broadcast following standard broadcasting algorithms. Blue arrows represent the return of RSSI data frames for each link. A link is defined as a physical level communication link between two wireless 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 RSSI value, a timestamp and a time-time-to-live value.

Figure 4.11 shows a simplified version of the algorithm that runs 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).getRssi() / mtrRatio;
    sigStrX = tlinks.elementAt(1).getRssi() / mtrRatio;
    sigStrB = tlinks.elementAt(2).getRssi() / 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 referring to each physical links 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 `Clinks` 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, the 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 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. After selecting the best three candidates for trilateration, the RSSI is converted using the Map-to-Real ratio. `Trilateration` is an abstract class that implements the trilateration algorithm. The internals of this class is defined as 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.

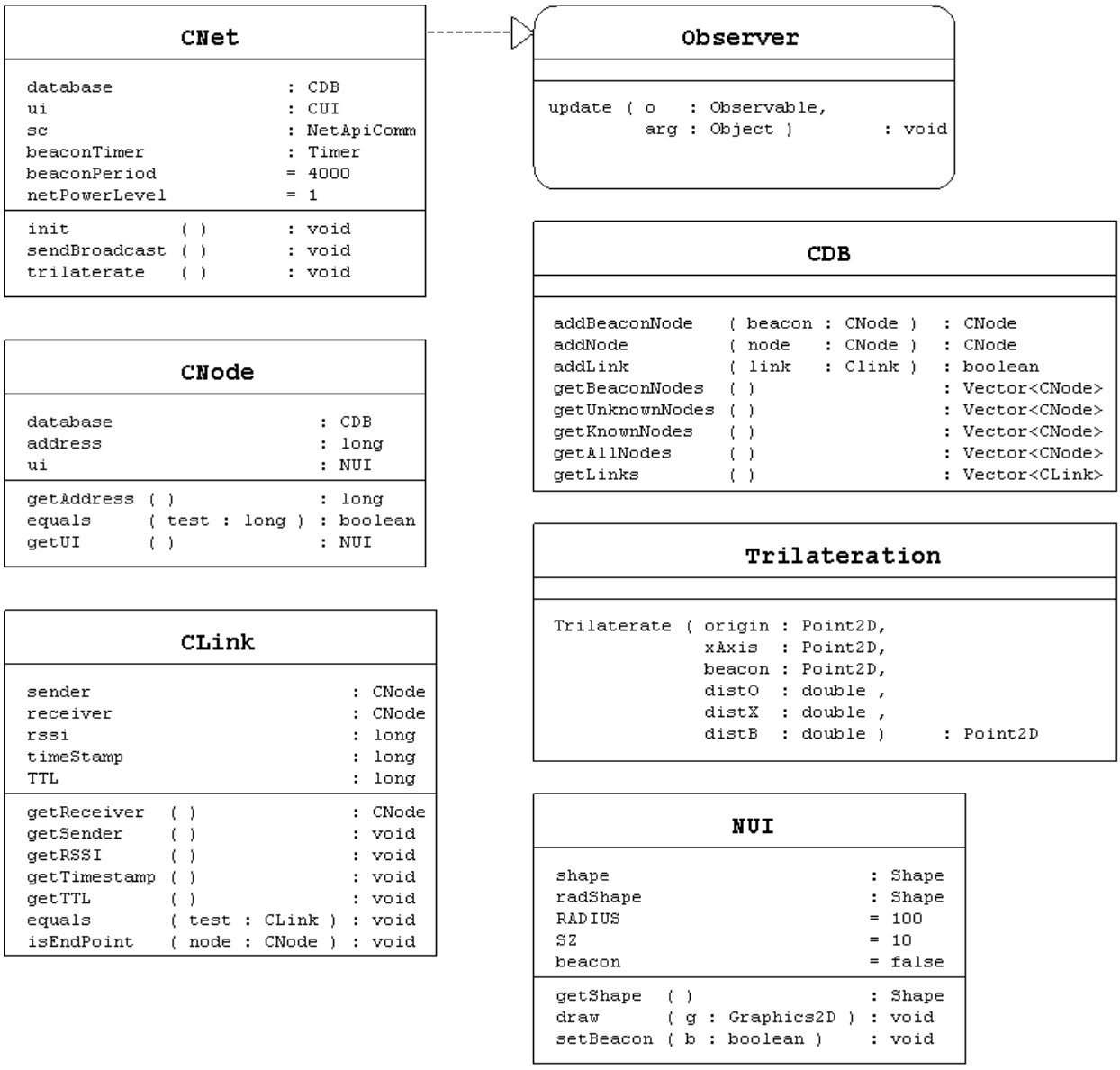

Figure 4.12 Central Server Simplified Class Diagram

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 an extremely 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. 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 its lowest power level. 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.

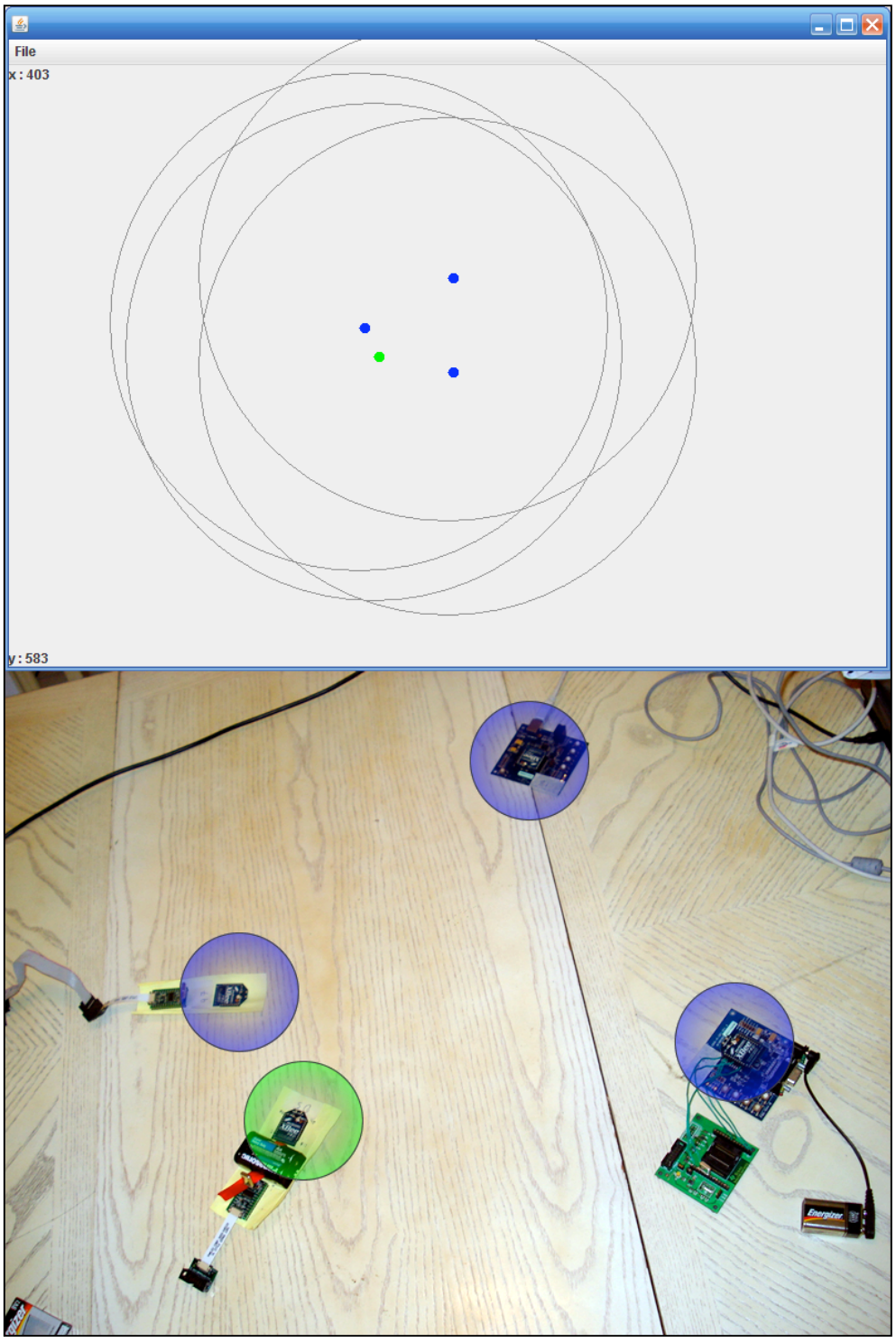

Figure 4.13 RTLS Prototype

Real-Time Location Systems are dynamic in nature. This dynamic inherent property cannot be observed in Figure 4.13, but our implementation does locate the nodes as they are moving. 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.

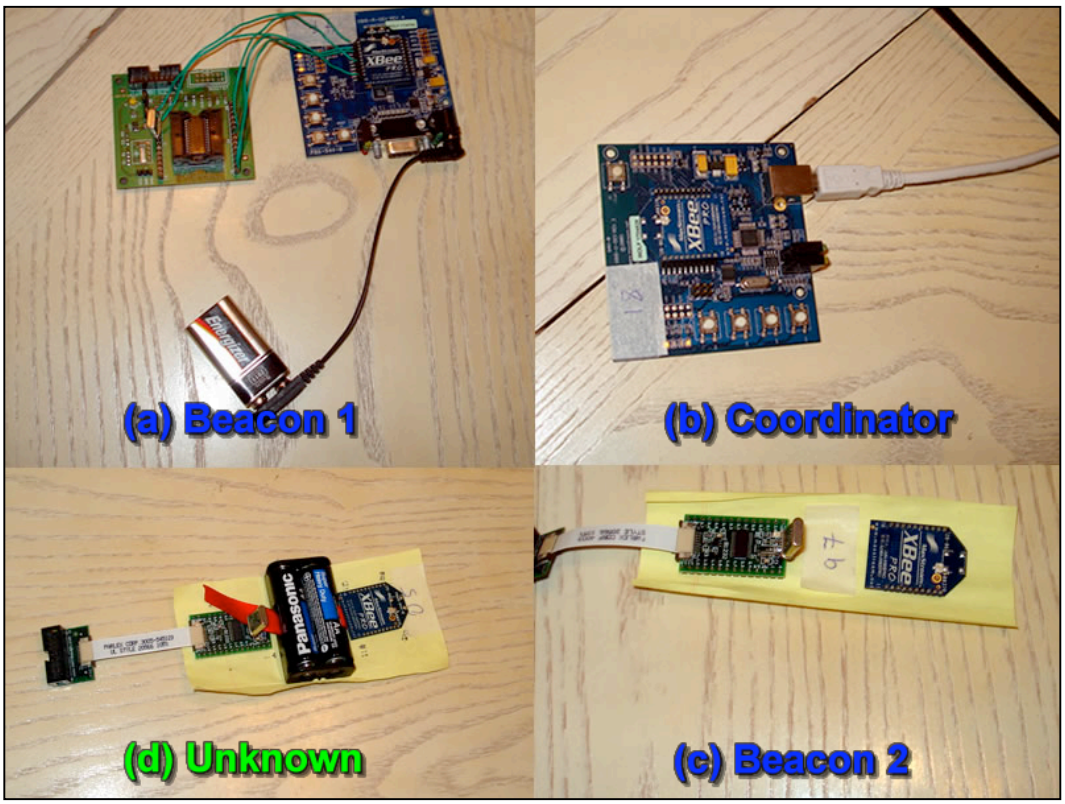

Figure 4.14 Wireless Nodes. (a) Beacon 1, (b) Coordinator, (c) Beacon 2, d) Unknown

In Figure 4.14 we can have a closer look on 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.

5 EXPERIMENTAL ANALYSIS

5.1 Introduction

This chapter presents a working implementation of our proposed software middleware for real time location systems on wireless mesh networks. The experiments help validate our solutions and ideas, as well as serve as a demonstration 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 verify the software middleware as a functional system.
2. To verify the prototype deployment location capabilities.
3. To measure the effects of network size in the location error.

To test the first and second objective, a fixed number of nodes were deployed 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 to be relocated by the central server. Each location was calculated several times using different sets of data. This experiment is detailed on sections ahead. The next experiment was carried out using a simulator developed to test the algorithms. The simulator accepts several parameter regarding the network size, error thresholds, node positioning and number of beacons. We can, in a theoretical 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 C programming language and compiled with msp-gcc, a GCC toolchain for MSP430. Our network simulator was developed using 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.

Figure 4.13 shows a similar setup used in this experiment. It was composed of three wireless beacons arranged in an equilateral triangle. An additional wireless node was used as the subject for location. Figure 5.1 illustrates the positions and distance of 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.

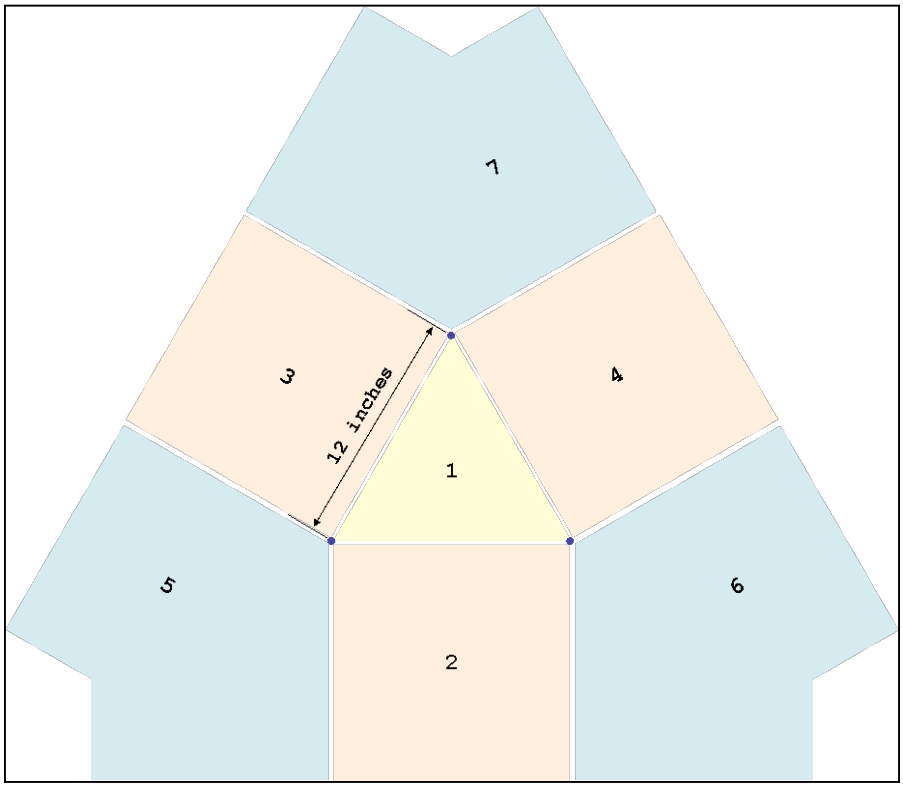

Figure 5.1 Experimental antennas setting

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 at the map

of the nodes. Figure 4.13 shows a similar setting and result. 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 at section 4.2 which is about 40 inches.

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

Figure 5.2 shows the error calculated for each individual measurement as described. 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%.

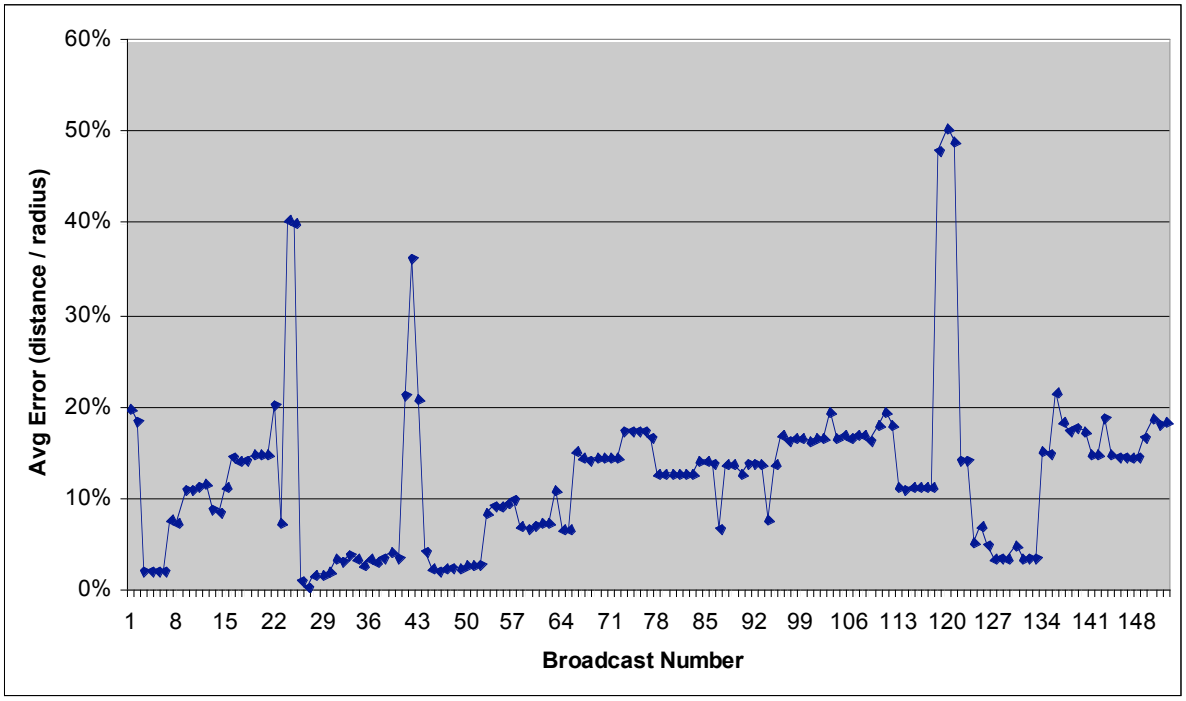

Figure 5.2 Position error at different broadcasts measurements

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 most applications 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 trough the network when we increase the network size. This experiment was carried out by a simulator. We start by briefly describing our simulation environment.

5.3.1 Simulator

The simulator environment is shown in Figure 5.3. It is similar to our prototype as 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 three nodes with a known fixed location and one of them starts the broadcasts, just as the 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 RSSI 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.

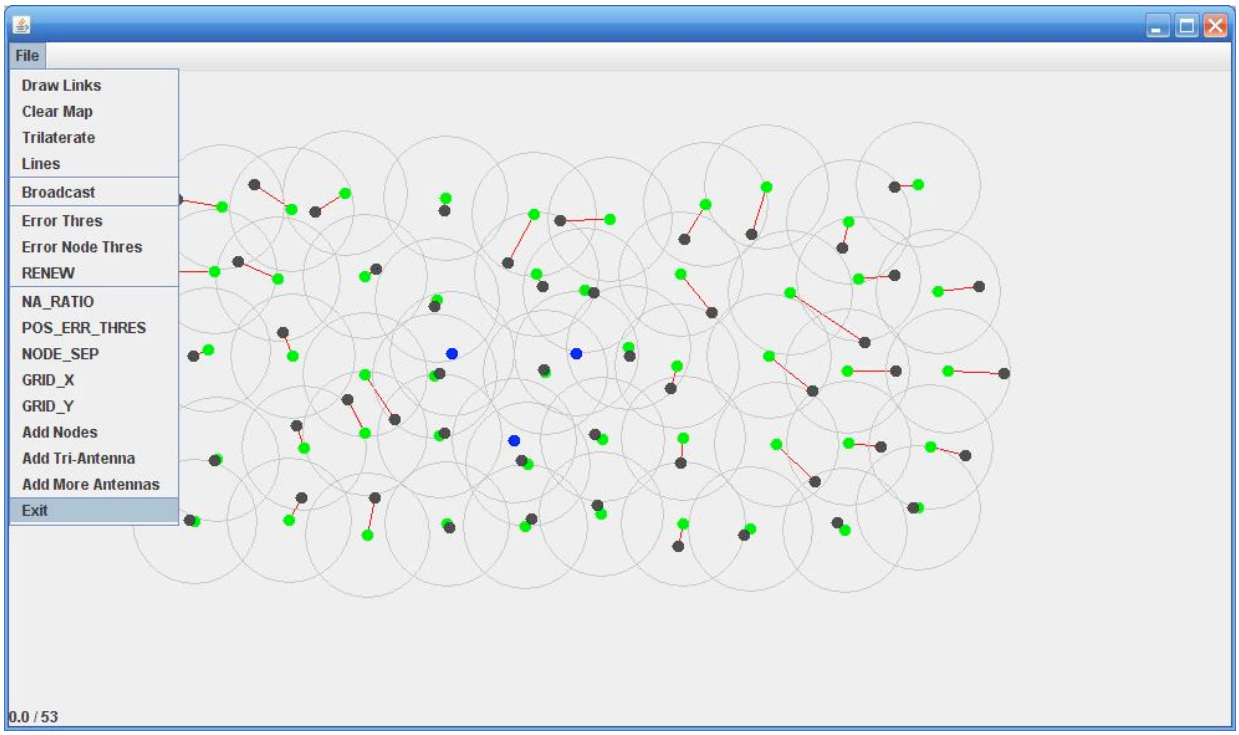

Figure 5.3 Simulation Environment

The 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 RSSI 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.

5.3.2 Distributed Location Error Propagation

This second experiment is used to determine how network size affects the average position error obtained. Previous section described the simulator's user interface. We did not used the user interface for this experiment, but used an internal programming interface to achieve the same simulation environment without the graphical overload.

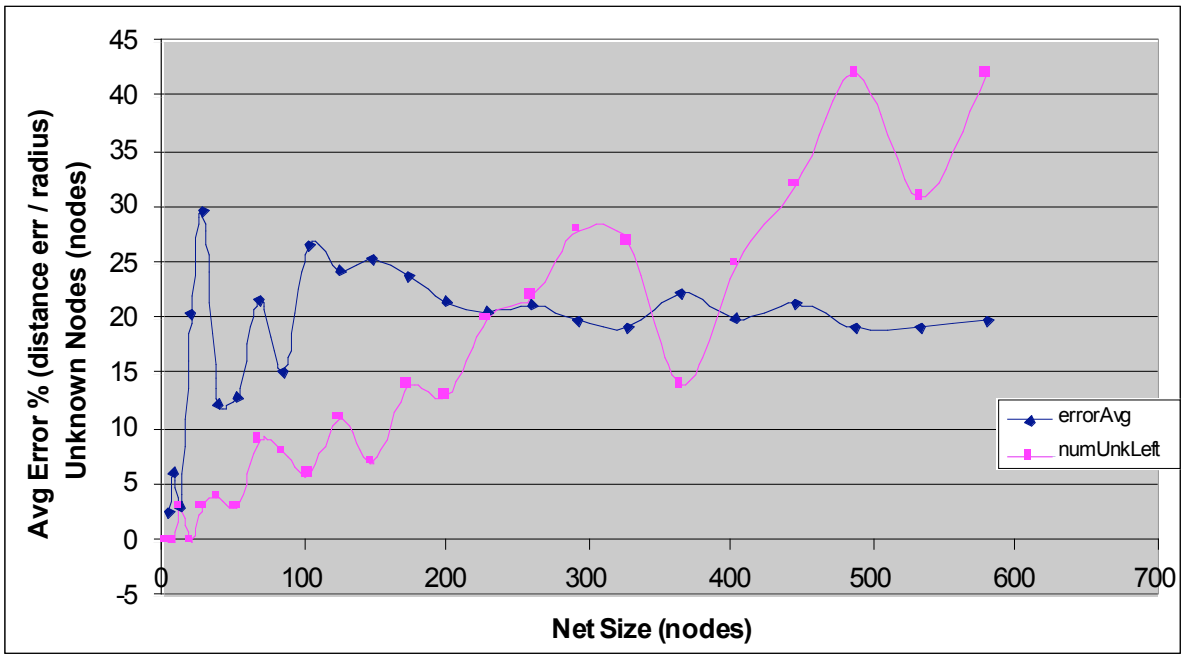

Figure 5.4 Average position error by network size

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 RSSI 12%
errThresNode = 12;
```

Then we iterated these parameters with several network sizes and obtained the results depicted in Figure 5.4. Notice however 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. The graph also shows another line which displays the amount of nodes that are not possible to locate. We can observe that it linearly increments with the size of the network, this is mostly the nodes at the edges of the network.

6 CONCLUSIONS

In this thesis we presented a software middleware for real time location systems on wireless mesh networks. This software system builds 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 an RTLS solution and numerous applications as well as future markets. The system architecture and mathematical procedures were described to some detail. We presented a working prototype implementation of the system as well as 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.

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. Fist 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 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 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.

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 RSSI 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.

Following we 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 the RSSI and the 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, as well as serve as a demonstration of the system. The experiments helped us verify 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 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.

6.2 Future Work

This section gives directions and suggestions for future developments concerning our work.

- *Wireless Nodes* – Wireless nodes firmware could be greatly improved on several areas regarding power management, communication speed improvements, RSSI sensing precision, calculation speed improvements, parallel RSSI 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 Hardware* – An elegant prototype package with minimal size, antenna protection, coin battery, weather protection, and external port availability.
- *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 RSSI data collection are a major improvement to data collection.

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 RSSI 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.