Delft University of Technology
Master's Thesis in Embedded Systems

A Mobile Inspecting Tool for Wireless Sensor Networks

Dan Avramescu

embedded software

TU Delft Delft University of Technology

A Mobile Inspecting Tool for Wireless Sensor Networks

Master's Thesis in Computer Science

Embedded Software Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Dan Avramescu
davramescu@student.tudelft.nl

19th October 2012

Author
Dan Avramescu (davramescu@student.tudelft.nl)
Title
A Mobile Inspecting Tool for Wireless Sensor Networks
MSc presentation
31.10.2012

Graduation Committee
Prof. Koen Langendoen, Professor, Embedded Software, EWI
Dr. Stefan Dulman, Assistant Professor, Embedded Software, EWI
Dr. Alexandru Iosup, Assistant Professor, Parallel and Distributed Systems, EWI

Abstract

The number of wireless sensor networks is constantly increasing. In the deployment phase of the WSNs projects it is common for failures to appear. These can be caused by software or hardware faults or by environmental factors. In order to find the source of these problems, the WSN needs to be inspected. Since the deployments of WSN are often made in inaccessible or hazardous area, the inspection operation can have a high degree of risk. In order to facilitate these intervention, the design and implementation of a WSN mobile monitoring tool is proposed.The tool is composed of an application built for a Neo FreeRunner smartphone which interfaces a TelosB mote via USB. The TelosB mote acts a sniffer and sends the data to the smartphone which analyzes it and displays packet information. The system is also able to use passive or active monitoring and track other nodes using a method of estimating distances to nodes using the RSSI value. The tool can be also used as a debugger for WSNs by replaying the last session and setting breakpoints. To evaluate this prototype, the localizing method is tested, together with the capacity of the sniffer mote process packets and the collision detection accuracy.

Preface

This Master thesis summarizes the work I made in the Department of Computer, Control, and Management Engineering Antonio Ruberti at Sapienza University of Rome, Italy, for my final project as a student. I have always believed that constantly taking challenges which will take me out of my comfort zone will repay me in unexpected ways. Coming from a bachelor background of Automation and Control, I was driven by a natural curiosity to step outside of my sphere and I decided to continue with a Master in Embedded Systems. Further, choosing the Embedded Software Group for my project allowed me to work with novel technologies and for the first time I felt like a real researcher.

This moment marks the end of a six year university period. Looking back at these student years, I am astonished by the amount of knowledge that I gained, all the student organization projects in which I involved myself and the great number of friends which I made. I am very grateful that I was given the opportunities to live in three different countries, with very different culture, an immense experience which has opened my mind and changed my perspective of life.

My project, like any engineering project, has had countless ups and downs. Luckily, I was supported by a group of people who helped me turn the downs into ups. For this I would like to thank Stefan and Andrea, my supervisors, who fed me with ideas and gave me guidance when I needed it. I would also like to thank Ugo and Francesco for all the help and advices they gave me for my project. Last, but definitely not least, I would like to thank my family, without who I would not have been able to come so far, for all the love and for always trusting in my decisions.

Dan Avramescu

Delft, The Netherlands
19th October 2012

Contents

Chapter 1

Introduction

1.1 Motivation

In recent years, the development of low power RF technology and microelectronics made it possible for remarkable results to be achieved in the direction of Wireless Sensor Networks (WSNs). The applications of these sensor networks are numerous, ranging from monitoring military missions, environmental conditions of a habitat, seismic activity, traffic to social interaction and many others.

In a typical WSN project, after the system has passed all laboratory tests the deployment phase follows. In this phase, the sensor network is set up in a real-world environment. Because of the wide range of applications, large wireless sensor networks are often deployed in areas with hostile terrain, no infrastructure support or environments which are very sensitive to factors such as the presence of humans. As described in [7], in most scenarios, deployment is a particularly challenging step, because apart from the difficulties arose by the terrain, bugs or unexpected performance losses appear due to environmental influences.

These faults appear as a results of the strong influences the environmental factors have upon the functions of the sensor motes. The laboratory tests can be conducted only on models which simulate the real world factors up to a certain extent, thus, unexpected behaviors in the quality of the communication links or even physical influences over the deployed nodes often appear. Some of the problems of various experiments have been pointed out in [7]. In a pioneer famous work [20] sensor nodes were deployed near Leach's Storm Petrels nesting areas for the observation of the birds nesting behavior. This experiment was subjected to node, network and data level failures caused by water infiltrating the sensors, false readings because of hardware failures, faulty temperature reading because of the inappropriate casing of the nodes and down times of the base station. In another experiment, [24], several nodes were installed 70m deep in a glacier in Norway with the purpose to study how climate change affects glacier dynamics. The authors speculate that more than half of the data was lost due to nodes being

moved out the transmission range by glacier movement, nodes have been physically damaged by the stress caused by the ice and last but not least, a software fault in the synchronization policy might have resulted in erroneous data. Another experiment which dealt with major deployment problems in its initial phases was the LOFAR-agro project [18]. To mention just a few items from the list of encountered problems in this case, an unfortunate chain of software related errors led to high network congestion and the depletion of the batteries in just four days and a inaccuracy in the routing mechanism caused most nodes not to send packets to the gateway. The "A line in the sand" project [5], consisted in a dense network of sensor nodes that were used for object detection. Because of the high density of nodes, the network became saturated and congested, a lot of data being lost due to collisions. A different project presented in [25] aimed to monitor the structural health of buildings by measuring seismic motion. However, the usage of an 8-bit counter which experienced overruns caused a lot of packets not to be delivered. Pipenet [32] is a network of sensors which was deployed with the purpose of monitoring pipeline infrastructures. One of the main problems experienced with Pipenet was the short-term missing data generated by an automatic watch-dog rebooting of the gateway at every midnight. In [36] a sensor network was deployed on a volcano in Ecuador with the purpose to measure seismic activity in the area. A software fault caused the time synchronization mechanism to fail leading to massive network congestion and packet collisions. Another project for environmental monitoring, Luster [30], has suffered from hardware faults such as bad contacts to the sensor and a malfunctioning storage node.

Situations like these are common to any project which aims to achieve a finished product. However, the work invested by researchers to find the root of the problems could use some facilitation. Many WSNs are equipped with a base station that stores the information and can, in some cases, even send it to a research facility for analysis. But in some situations, the base station can also be malfunctioning or suffer from an outage [18][24][36]. Moreover, with recent development, a lot of projects implement Ad-hoc Networks or even Mobile Ad-hoc Networks(MANET) which do not use a sink node to send data to, but rather collect and distribute data in a decentralized manner. In this cases human intervention for the retrieval of information about what is causing the system failure and the retrieval of nodes is mandatory and it can sometimes imply risks.

Consider a scenario where a WSN which is deployed in a very rough terrain (e.g. a deep jungle or a volcano crater) is malfunctioning. Thus, in order to regain functionality of the WSN, a technician needs to be sent over to investigate what is causing the failures. Since the terrain does not allow the space to handle a laptop, a specific terminal device needs to be used in order to retrieve the data needed for analysis. Literature exists on such devices, the project described in [11] being a relevant example.

The development of terminals on specific platforms has proven to be a viable solution for investigating problems that appear in WSNs. However, parallel development of smartphones and of Cloud Computing in the past years have transformed the first mentioned from the classic devices used to share information verbally and by means of text, into terminals for a large number of applications. Since the new generation of smartphones provide mobility, storage and great computational power, this can be used as a great advantage in different types of industrial applications. This has naturally encouraged a migration from the classic monitoring and interaction devices to applications for mobile general purpose platforms.

This thesis proposes to build a tool which provides real-time monitoring as well as logging capabilities of the packets sent through a wireless sensor network. The application can be described as a packet sniffer for WSN which incorporates several features which will be described in a later chapter. This application is to be built on top of a system made out of a smartphone and an off-the-shelf mote capable of interfacing the wireless sensor network communication channel. By using a smartphone instead of a dedicated terminal, technicians and researchers can operate on multiple networks with a single device, in spite of the different protocols or frequencies which are used by the networks.

1.2 Research Questions

As stated, the goal of this thesis was to construct a smartphone application that is able to intercept and analyze packets from a WSN and provide both a graphical interface to display the acquired information for on the spot observations and a data logging feature, for later analysis. In addition to this, the application should be able to act as a packet injector in order to introduce user defined packets into the network stream, with the purpose of observing more scenarios of the behavior of the WSN. Moreover, in the purpose of on-location analysis of the motes, the application should be able to also indicate the location of a node within an acceptable area, to facilitate operations on the actual node. Three research questions will be presented in the sections below, and each of them will be answered in the following chapters.

1.2.1 Interfacing the smartphone with a WSN

The most used sensor motes available on the market operate using the IEEE 802.15.4 WPAN technology [13]. In contrast, none of the available smartphones have support for the IEEE 802.15.4 technology. Naturally, the first research question that arises is related to establishing a communication channel between the smartphone and the WSNs. Which can be the common channel?

1.2.2 Locating the nodes

The second research question that needs to be answered for the completion of this project is related to the scenarios in which the application will be used to actually locate the nodes, for further operations. Thus, the question is, how to manage to indicate the location of a node in *any kind* of network topology, using just the application?

1.2.3 Application performance

Once the application is running, its performance needs to be evaluated. The following questions need to be answered in order to validate the initial goal of the project:

- What is the maximum frequency of packet arrival that can be detected?
- Does this change in the case when the packet injector is enabled?
- How can the retrieval of inconclusive data be avoided?
- How can the application correctly indicate the status of the network?

1.3 Outline

The present work is composed of seven more chapters. Chapter 2 presents background information on the most important fields which the thesis relies on along with a survey on the work related to the topic of the thesis. Chapter 3 discusses the hardware choices which were made for this project. The next Chapter, 4, talks about the design choices and the software implementation of the application. The node localization technique which was used in the application is presented in Chapter 5. The evaluation of the prototype is described in Chapter 6. Chapter 7 represents a series of discussions about the challenges which have risen during the implementation phase of the project. Finally, Chapter 8 concludes this thesis and presents a number of ideas for future work.

Chapter 2

Background

This chapter will briefly discuss the domains which have served as a starting point for the topic of the present thesis. Section 2.1 is a brief introduction in the Wireless Sensor Network field, while 2.2 presents some details about smartphone devices. Section 2.3 is the result of a literature survey regarding projects that involve smartphone applications which interact with WSNs, localization of nodes in WSNs using simple mechanisms such as the Received Signal Strength Indicator (RSSI) reading of packets, and software solutions similar to the one proposed for the goal of the current project.

2.1 Wireless sensor networks

This section will give a brief description of the rather vast Wireless Sensor Networks field. A very comprehensible paper, [31], was the main source for the structure of the description, and a number of quotes, which will follow in the next paragraphs.

Introduction to WSN

Wireless sensor networks were born out of the idea that with a large number of small individual sensor nodes with limited features, the strength of the entire network will be sufficient for the desired purpose. A WSN is composed out of routers, gateways and of course nodes. These components cooperate to transmit the sensed data from their neighboring environment to the user.

Even if as early as 1950s the term of wireless sensor networks was mentioned, the moment when the first wireless local area network protocol, the 802.11, was introduced, in 1997, marks the beginning of the modern sensor networks. The 802.11 was soon upgraded to 802.11b with a higher bandwidth and mechanisms for medium access control (MAC) such as the Carrier Sense Multi Access with Collision Avoidance (CSMA/CA).

WSNs follow Moore's Law towards miniaturization of processing devices. Busi-

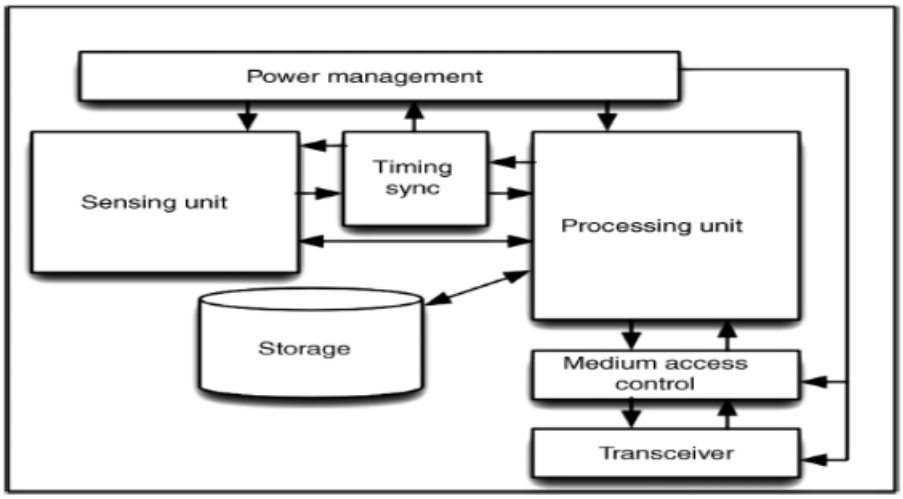

Figure 2.1: A generic sensor node architecture,[31]

ness Week stated in September 1999 that WSNs will be the technology with the most impact over the 21st century. Further more, the publication Technology Review of MIT claimed that WSNs are one of the top ten emerging technology.

The smallest individual device which is included in a wireless sensor network is called a node. Between themselves, the nodes communicate in a wireless manner. It is common for for nodes to self-organize themselves after deployment, in which case we are discussing about an ad-hoc WSN. In the design of WSNs projects, five important characteristics need to be treated with maximum care: scalability, reliability, security, self-healing and robustness.

Sensor nodes

Sensor nodes are, as stated, the small devices which can read sensory information, can perform processing operations and can communicate with other sensor nodes. The architecture of a node. as shown in Figure 2.1, is made up of several units. The sensing unit is responsible for detecting environmental parameters (e.g. humidity); the power management unit provides power for the device while in the same time controls the energy consumption in order to achieve energy efficiency; the processing unit handles computations and communications; the storage component works with the processing unit to handle recorded data and the transceiver unit interacts with a MAC component to communicate with other nodes.

The protocol stack

[4] describes very well the protocol stack. The protocol stack promotes power and routing management, communicates through the wireless medium, encourages

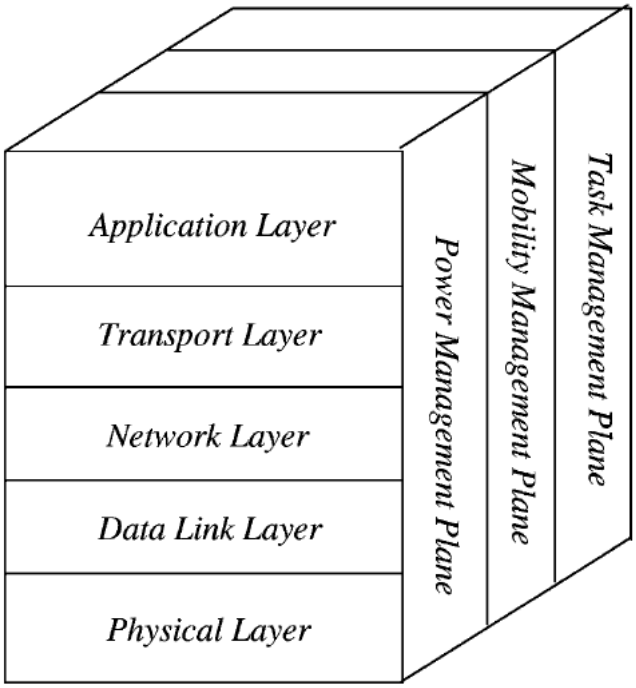

Figure 2.2: A generic sensor node architecture,[4]

cooperative tasks of the sensor nodes and integrates data within networking protocols. A representation of the stack is shown in Figure 2.2. The components of the protocol stack are the five layers, application, transport, network, data link and physical layers, and the three management planes for tasks, mobility and power. The application layer holds software applications for sensing tasks; the transport layer maintains the data flow between the sensors; the network layer routes the data through the network; the data link layer uses the MAC protocol to assure a collision free transmission and the physical layer deals with transmission and receiving techniques. As for the planes, the power management plane manages how the node uses its power to interact with the network using minimum energy; the mobility management plane maintains the transmission paths regardless of movements of sensors and the task management plane schedules the sensing tasks within a region where not all the nodes need to perform the same task at once.

Standards and Specifications

In order to assure a certain level of standardization, there are a number of authorities in the field of WSNs such as IEEE (for physical and MAC layers), The Internet Engineering Task Force (layer 3 and above) and The International Society of Au-

tomation (vertical solutions for all protocol layers). The common standards which are used in WSN regularly are: Wireless HART; ISA100; IEEE 1451; ZigBee / 802.15.4.

IEEE 1451 In 1993, IEEE and National Institute of Standards and Technology set up the IEEE 1451. the Standard for Smart Sensor Networks. The objective of the IEEE 1451 standard is to ease the efforts of producer of sensors to develop smart sensors which can interface networks. IEEE 1451 was introduced because it would be infeasible for smart sensor developers to make transducers for every type of network available.

IEEE 802.15.4 The IEEE 802.15.4 (or ZigBee) protocols have become the base for many WSN system today, due to their low-power consumption. The Physical and Medium Access Control layers of the networking model are defined in the IEEE 802.15.4 protocol. This enables communication in the 868-915 MHz and the 2.4GHz ISM bands with data rates of up to 250kb/s. Furthermore, ZigBee adds a great number of features over the 802.15.4 layers such as : longer data transmission through mesh networking topologies, interoperability with other devices, security, customization and flexibility of the protocol and used-defined application objects.

In 802.15.4 the MAC protocol introduces the notion of superframe structure. The interval between two beacons, which are coordinating packets, delimit the superframe. Through beacons the WPANs synchronize and offer information about the medium. There is also an alternative of WPANs without beacons, where all nodes adopt a non slotted CSMA/CA scheme. However, in a superframe, sensor nodes can be inactive or active, in which case the active period can also be split between a contention period and a contention-free period. In this case, the contention period is slotted and nodes use CSMA/CA to access the channel in every slot.

Operating Systems for WSNs

As WSN applications are built with specific purposes upon an architecture of low power nodes, the Operating Systems for WSN are similar with the OSs of Embedded Systems.
The first OS which was built exclusively for WSNs is **TinyOS**. TinyOS applications are based on components, which connect between themselves via interfaces. The applications are written in the nesC language, which derives from C. Components and interfaces include packet communication, storage, sensing, actuation and routing abstractions.

TinyOS is an event driven based OS, which means that applications constitute out of event handlers and tasks. In the case that an event is detected, the OS signals a specific event handle which in turn can post tasks. Tasks are scheduled

for later execution by the kernel.

2.2 Smartphones

The early mobile phones, or cell phones, are devices which through a radio link allow the user to exchange information in various forms, such as voice or text, regardless of their position. To access the telephone network, the mobile phone connects to a cellular network of one of the mobile phone operators.

Throughout their development, mobile phones have grown to offer support to a range of more advanced features such as MMS, email, wireless communication (infrared, Bluetooth, IEEE 802.11), and Internet access. Following the decrease in size of microprocessors, a consequence of Moore's law, more powerful processing units could be equipped on mobile phones, allowing them to have their own operating systems, more connectivity and more computing power. This transformed mobile phones into what is generally known as *smartphones*.

Smartphones were gradually added support for various types of equipment such as GPS modules, digital cameras, media player support, touchscreens, and Internet connectivity provided through either Wi-Fi or Mobile Broadband. As they posses a general purpose operating system, a large variety of applications can now be installed on smartphones. The next section will describe the most common operating systems for smartphones.

Operating Systems for Smartphones

There is a large number of operating systems for smartphones available on the market. In spite of Android from Google and iOS from Apple currently being the dominat players, other alternatives are finding their place on the market.

Android Android is an operating system intended for touchscreen smartphones. It is developed and maintained by Google in association with the Open Handset Alliance. Android is a Linux-based OS. The source code is available under free and open source software license. The Linux kernel changes are released under the GNU General Public License version 2, while the rest of the source code is published under the Apache License version 2.0.

A large community of developers are contributing to the Android System functionality by writing applications for it. The applications can be developed using the Android Software Development Kit, in Java, or with the Native Development Kit in C or C++. Other frameworks used for developing Android apps include Google App Inventor or different cross platform solutions.

iOS iOS is a mobile OS developed and distributed by Apple Inc. iOS is destined to run on Apple devices only. Apple were the first to introduce the user interface based on direct manipulation through multi-touch gestures. Swipes, taps, pinches

and similar gestures represent the novel interaction with the OS.
Previously, Xcode, an IDE equipped with a software development kit was released in order to allow third-party developers to construct OS X applications. At the release of Xcode 3.1, the development tools for the iOS SDK were included and third-party developers were able to write apps for iOS. The iOS apps are written in Objective-C.

Symbian Symbian is an operating system for smartphones which was developed by Nokia and later passed on for maintenance to Accenture.

Unlike the early Symbian OS developed by Nokia, its successor, the Symbian platform includes a user interface.

The main SDK of Symbian has been since 2010 the standard C++ with Qt. Applications for Symbian are usually created with Qt Creator or Carbide.c++. Alternatively, Python, Adobe Flash Lite or Java ME can be used for developing applications.

BlackBerry OS Research in Motion (RIM) has developed the BlackBerry OS for their own BlackBerry type of smartphones. The BlackBerry OS supports multitasking and it offers as a native system a comprehensible support for corporate email through MIDP 1.0 (and recently, a subset of MIDP 2.0).

Developers of BlackBerry software can write applications using the BlackBerry API, keeping in mind that applications which use specific functionalities need a digital signature.

Windows Phone Microsoft has its own OS for smartphones, the Windows Phone OS. It is targeted for the consumer market and it was lunched in 2010. A new user interface was created for Windows Phone which implements a design language named Metro.

Applications and games for Windows Phone are based on Microsoft XNA or a version of Microsoft Silverlight. These apps can be designed and tested in Visual Studio 2010 which provides the Windows Phone Developer Tools extension. The future generation of the OS, Windows Phone 8 will together run managed code through a Common Language Runtime similar to the one that the Windows OS is using, which tighter with the support for C and C++ libraries will allow Windows applications to be ported on the Windows Phone OS.

Open-source development Apart from the branded and open source OSs presented above, the open source community has developed both open source hardware and software for smartphones. An example of such projects is OpenMoko. OpenMoko is a project intended to develop open source smartphones and OSs. One of the first projects of OpenMoko was the OpenMoko Linux, a OS based on Linux for their family of devices. The OpenMoko platform will be discussed further, later in this thesis.

2.3 Related Work

2.3.1 Interfacing nodes with smartphones

A system for collecting and forwarding over the Internet real-time sensor data through a smartphone was described in paper [40]. This experiment sampled an accelerometer, a temperature sensor and ECG signals through a device equipped with a MSP430BT5190 Mixed Signal Controller which communicates via UART with a CC2560 Bluetooth transmitter. The Bluetooth transmitter was then used to forward the data to the smartphones which could in turn forward it through the Internet. The system proves that smartphones can interact with WSNs, although the interfacing is done through a self developed device, rather than with an off-the-shelf solution.

The authors of [15] present a project which bridges an Android smartphone and motes running TinyOS. Their application Practical Body Networking (PBN), is designed to enable practical activity recognition on a mobile device. The sensing system behind PBN is represented by IRIS motes which are placed on the human body. The communication with the Android device is made via USB with a TelosB mote, used as base station.

One of the innovation of this topic consisted in enabling the Android smartphone to support external USB devices, considering that the USB host controller documentation of Google was not publicly available. This has been overcome by modifying the Freescale MPC5121 driver in order for it to function with the chipset of the smartphone, thus enabling USB device recognition. Since the Enhanced Host Controller Interface (EHCI) controller of the smartphone did not provide power for external devices, a battery pack was attached to the TelosB node.

To enable TinyOS support on the Android platform, the PBN team took advantage of the fact that each mote had a FTDI USB serial chip for communication with the host. In order for the Android platform to provide the same, the device manager was modified to establish correct privileges for the FTDI driver, which is responsible for creating a serial port device.

A more advanced project, MobiCon, has been developed since 2007 until recently, and it is presented in [19]. MobiCon is a middleware system that is meant to interface OSs of mobile and sensor platforms.

The most important component of MobiCon is a sensor broker which negotiates the communication between the mobile platform and the sensors. In order to achieve this communication, the MobiCon team has developed protocols for data reporting, sensor control messages and sensor detection. Common interfaces and a message protocol were also integrated into this component to allow independence for interacting with diverse sensor platforms.

This is a very promising project for mobile platforms-WSN interaction, the only downsize to it being that the connection between the smartphone and the sensor

was made with a Bluetooth-to-serial converter which was physically connected to the sensors. For the purpose of this thesis, such a solution is not preferred as the requirement states that the position of the sampled nodes is not necessarily known, and one of the features of the project is to actually locate the nodes within a certain area.

2.3.2 Localization of nodes

Much research effort has been put into developing localization methods for WSNs [21], [12], [26]. To classify them, these algorithm can be split into two categories: range free and range based localization schemes.

Range free localization schemes do not use metrics as point-to-point distance estimations or angle information to calculate the position of nodes. Instead, these methods use relative positions to localize nodes. The DV-hop algorithm [23] is a basic range-free localization method which has inspired other research works such as [37],[14],[41],[6] and countless others. DV-hop uses a distance vector-hop exchange so that all the nodes in the network receive distances to node beacons with known locations in hops instead of distance metrics.

The range based localization schemes use point-to-point distance or angle information to compute node locations. The range based localization methods are Time of Arrival (TOA), Time Difference of Arrival (TDOA), Angle of Arrival (AOA), and Received Signal Strength(RSS).

TOA algorithms measure the arrival time of the signal at the receiver and computes the distance based on the transmission time [38]. TOA methods need perfect time synchronization between the nodes in order to return an accurate distance estimation.

TDOA is a version of TOA where time synchronization is not needed [33]. On the other hand, an additional transceiver of different nature, such as an ultrasound transceiver [22] has to be equipped on nodes. For measuring the distance, a node sends both RF and ultrasound signals simultaneous. The receiver calculates the distance from the transmitter by measuring the difference in time of arrival of the two signals with known propagation speeds. The downside to this localization method is obviously the need of additional hardware on the nodes.

AOA methods measure the arrival angle of the received signal from a node with a known position [34], [8]. In this case the beacon nodes have to use a special antenna which sends omni-directional signals. This method is unsuitable for the purpose of this project due to the necessity of beacons. RSSI localization methods make use of the RSS of packets which were received. The benefit of Tthese algorithms is that they require no time synchronization between nodes and no additional circuits or equipments.

For the localization feature of the monitoring tool a simple mechanism which

will approximate the distance to a node must be used. The methods based on RSSI reading value are preferred, as the projects focuses on appreciating the distance with the least amount of additional equipment as possible.

A localization algorithm, PBM, based on RSSI and the Bayesian principle is presented in [9]. The algorithm aims to reduce the measurement noise influence in order to increase localization accuracy by using statistical methods. It also includes the usage of beacon nodes - nodes with known positions - to divide the deployment region of the WSN into small grids. Further, it uses the confidence level of grids with known position to determine the positions of unknown nodes. Although this algorithms provides good accuracy, the necessity of beacon nodes is not compliant with the requirements of the topic of this thesis.

Paper [29] makes a study on how accurate is the RSSI model in a WSN for approximating the location of a node. They have made two types of observation: calibration-based analysis and a location estimation scheme : the k-nearest signal space neighbor match algorithm. Out of the two, the first one is of interest for the purpose of this thesis.

With the calibration-based analysis, the authors pursued the confirmation of their theoretical analysis that the distance is an exponential function of RSSI. Experiments were conducted in worst-case, a laboratory environment, and best-case scenarios, an open space, and they could conclude that the relation between the RSSI and the distance can be represented with a linear curve fit. Even though the results were prone to a certain degree of error, the error tends to shrink considerably when switching from a laboratory environment to an open space one.

Independent from paper [29], Kumar et. al. follow a similar experiment in [17]. The authors have used TelosB motes for measuring the RSSI values under *ideal* conditions. In these parameters, they managed to estimate the position of the nodes with an almost uniform error.

2.3.3 Wireless sensor sniffers

This section surveys various topics found in WSN sniffer applications which were implemented without the support of mobile platforms. Nevertheless, some of the examples below are very useful for real-time monitoring of WSNs.

In an early work, Yang et. al. describe in [39] a self-developed tool called SNAMP: Sensor Network Analysis and Management Platform. SNAMP is described as a multi-sniffer and multi-view visualization platform for WSNs. This system is built to collect information from a network made out of GAINS-4a nodes. For this purpose GAINS-3 nodes are used as sniffers, and are attached with serial-to-Ethernet cables to a visualization server.

The SNAMP software component integrates four modules for analyzing and

managing the WSN : a topology analysis module, a sensing data module, a packets analysis module and a network measurement module. The system also provides debugging functions such as the ability to record and replay network activities and setting up breakpoints in the packets view.

A tool called Sensor Network Distributed Sniffer (SNDS) is presented in [16].The main components of SNDS are a sniffer infrastructure and a service program for data analysis. The sniffer infrastructure is made out of a set of not complex network sniffers which communicate with each other through Ethernet. The sniffers are used to monitor a certain frequency and forward the acquired data to the service program. The main function of the the service program is to act as a WSN protocol analyzer.

Paper [10] conducts a study on the performance of of-the-shelf WSN motes which are used as sniffers. As a testbed, TelosB sensor node motes running TinyOS 2.1.0 are used as sniffers. The authors looked to answer a series of questions related to the performance of the mote as a sniffer. Their results, both analytical and empirical, indicated that a sniffer can monitor up to 60 packets per minute with little buffer overflow. The TelosB motes also proved to create very little delays on per-hop measurements (less than $300us$). Moreover, the per-hop loss measurements from sniffers did not differ from the ones of a receiver. The authors concluded that the quality of the measurements made with motes turned into sniffers is satisfactory.

A comprehensive and modern work named NSSN, a network monitoring and packet sniffing tool for wireless sensor networks, is presented in [42]. The architecture of NSSN is more vast, containing a monitoring network, a server, several remote monitor clients and the WSN which is to be observed. The monitoring network can be described as a path of many sniffer nodes which transfer the data to the monitor server. This transmission path can be either wired or wireless (3G, Wi-Fi, GPRS). The nodes which act as sniffers were developed by the Institute of Software of the Chinese Academy of Science. The listening module of a node can function over the 430M, 980M and 2.4GHz frequency bands and on each frequency the mote can identify automatically multiple network channels on which WSNs can be deployed. In this case, it is obvious that NSSN can automatically search all the 16 available channels of the IEEE 802.15.4 protocol and monitor each of them at the same time.

After NSSN has captured packets, it can be used to monitor the network status and find network problems by using its built-in features: protocol parsing and displaying, network diagnosis for performance measurement, data mining, network monitor and statistical analysis. Another important feature of NSSN is the development of a mobile prototype, which can be use for in-field work. This aspect is very similar to the goal of this thesis, with the exception that the NSSN is using a self-developed prototype instead of creating a smartphone application.

It is worth to mention in the final part of this chapter that the French company TazTag have claimed at the GSMA Mobile World Congress 2012 in Barcelona that they will release the first smartphone which supports amongst other, ZigBee connectivity. The smartphone is going to be equipped with an Android 2.3 OS, and a tablet platform will follow its release. However, the lunch of this device which was expected to be in March 2012 has been so far postponed. When the TazTag device will be lunched, it will open the way for the development of countless applications which will interact with ZigBee devices, which cover an area more vast than just that of WSNs.

Chapter 3

Hardware setup

A central role within the current project is played by the smartphone which will interact with the WSN. Even though many mobile platforms are theoretically available for testing such a project, one important challenge arises. While the smartphones can achieve connectivity in different forms, all the models of all the developers are so far missing a module which would allow them to communicate on the frequencies which are reserved for the 802.15.4 standard. For this project, in order to be able transmit the data from the WSN to the smartphone, an interconnection smartphone-WSN will be needed. It is because of this why an additional hardware device is required to perform the bridging between the smartphone and the WSN. The device which was chosen to offer 802.15.4 connectivity is a ZigBee sensor mote, as sensor motes offer native 802.15.4 radios and are widely available on the market for affordable price (less than 100 euros per mote). As the main task of this mote will be to forward packets which are collected from the WSN nodes within range, it will be from now on referred to as the ”sniffer mote”. The hardware choices for the implementation of this project were an OpenMoko Neo FreeRunner smartphone and a Crossbow TelosB sensor mote. A more detailed description of the two devices is given in the below sections, together with the reasons for which they were chosen.

Another issue that needs to be addressed is finding a method a way to establish a connection between the smartphone and the device which is able to interface the WSN. Several methods are proposed in the literature, most of them offer a Bluetooth interfacing method ([28],[19],[40]) or a USB connection ([15])to the smartphone.

3.1 The wireless sensor mote

It was decided that in order to obtain an application which will be flexible and affordable, the sniffer mote must be chosen from the motes available on market. A TelosB mote was chosen, as it presented immediate availability and an easy to

Figure 3.1: A TelosB mote

achieve USB connection with a smartphone. In addition to this, the description of the implementation from [15] also encouraged this choice. The TelosB mote platform (Figure 3.1) is an open source, low-power wireless sensor module. The TelosB offers USB programming capability, an IEEE 802.15.4 compliant, high data rate radio with integrated antenna, a low-power MCU with extended memory and an optional sensor suite [1]. The TelosB data sheet gives a more complete list of features for TelosB motes, as seen below:

- IEEE 802.15.4 compliant RF transceiver
- 2.4 to 2.4835 GHz, a globally compatible ISM band
- 250 kbps data rate
- Integrated On-board Antenna
- 8 MHz TI MSP430 microcontroller with 10kb RAM
- Temp / Humidity Sensor
- Light Sensor
- Low current consumption
- 1MB external flash for data logging
- Programming and data collection via USB
- Runs TinyOS 1.1.11 or higher

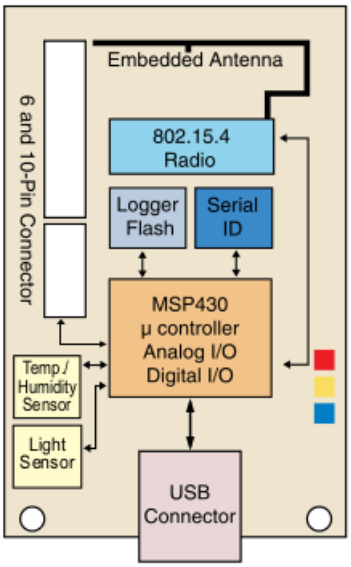

Figure 3.2: The block diagram of a TelosB mote

Interconnectivity As shown in Figure 3.2, in order to connect the TelosB mote with a smartphone and allow the transfer of data, a standard USB cable can be used. However, in the interest of the device to be recognized as a serial port with regard to the Windows environment, or to be mounted as a device in the /dev directory for operating systems such as Linux, OSX and BSD, it is necessary to install the Future Technology Devices International (FTDI) driver. This driver allows to use the USB connection exactly like a serial port. Once the mote is connected via USB, the system will create a port to which the mote is assigned, to be used for communications. Obviously, more than one mote can be connected at the same time, each of them being assigned to a different port. In order to use the serial communication for forwarding network packets, a TinyOS application which implements the $Serial802\_15\_4C$ component can be loaded onto the TelosB mote.

3.2 The smartphone

The smartphone which will be used in the implementation of the project is required to have USB host capabilities. The USB host mode allows connections of different USB devices, such as input devices, storage devices, or in this case, a sensor mote, to the smartphone just like a regular computer.

The preferred choice for the smartphone was a device running an Android op-

Figure 3.3: The Neo FreeRunner smartphone

erating system. However, due to the lack of Android devices available for testing and the availability of an OpenMoko platform, the latter was chosen for the implementation of the project.

3.2.1 OpenMoko

OpenMoko is a project aiming at developing smartphones with an open source software stack. The first smartphones to be released was the Neo 1973, followed by the Neo FreeRunner model. For OpenMoko smartphones, users have a large variety of open source operating system distribution that they can choose from.

The available OpenMoko model for developing and testing the WSN inspection tool was the Neo FreeRunner. The Neo FreeRunner is indeed intended for users with a high demand for customizability. It can be characterized as a Linux-based touch screen smartphone which is preferred by Linux users and software developers who appreciate the total freedom they have to use and design applications.

As OpenMoko distributions are based on Linux, there are a series of features which would point the Neo FreeRunner as a good choice for the implementation of this project:

Opensource From both hardware and software points of view it is completely open source.

Simple application development OpenMoko uses popular open source technologies such as GTK, X server or ALSA.

USB Host The Neo FreeRunner comes equipped with USB Host support, which is an obvious advantage for the implementation of this project.

The full list of specification for the FreeRunner is presented below.

- High resolution touch screen 2.84 (43mm x 58mm) 480x640 pixels
- 128MB SDRAM memory
- 256 MB integrated flash memory (expandable with microSD or microSDHC card)
- microSD slot supporting up to 16GB SDHC cards (Supported microSD cards)
- Internal GPS module
- Bluetooth
- 802.11 b/g Wi-Fi
- 400MHz ARM processor
- 2 3D accelerometers
- 2 LEDs illuminating buttons
- GSM and GPRS
- USB Host function with 500mA power

The Architecture of OpenMoko is, as it can be seen in Figure 3.4, based mostly on open source technologies.

Linux 2.6 kernel and drivers The project uses the vanilla kernel 2.6 series, updated with OpenMoko-specified updates. It includes support for embedded devices like USB, SD, touchscreen and communication drivers. The patch set includes additional support for various embedded devices like USB, SD, touch screen, and communication drivers. These drivers are loaded at boot time and they all reside in the kernel space.

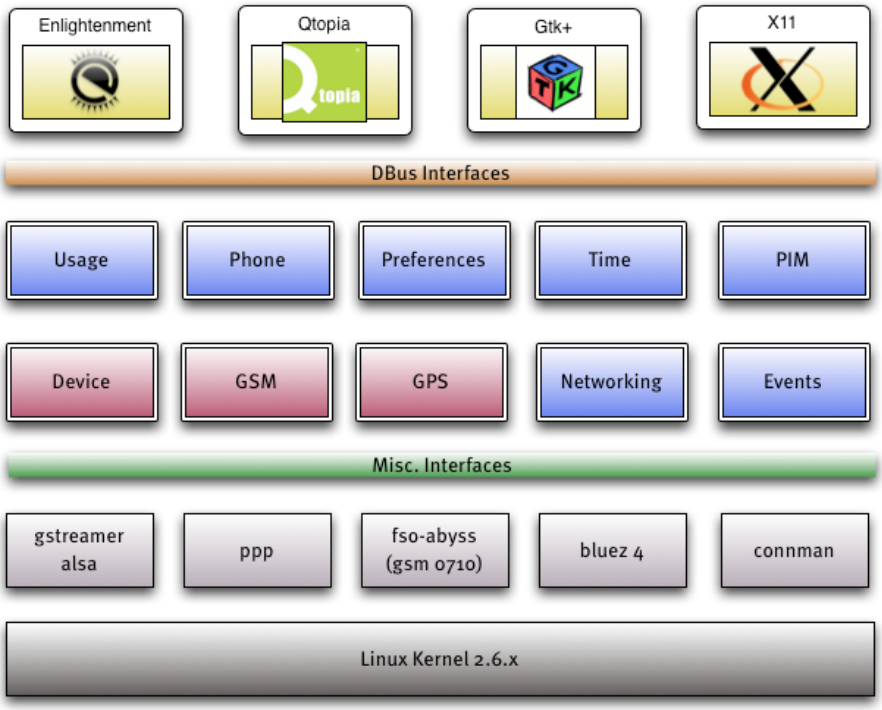

Figure 3.4: The OpenMoko architecture

Linux Core services udev: provides standard way for interaction between components and acts as the device manger and handles the device nodes in a pseudo /dev file system.
blueZ: is responsible for all the operations in OpenMoko which regard Bluetooth.
Dbus: Manages the communication between applications by providing an interprocess communication (IPC) GSM: It is composed of the core GSM module, GSMd (a daemon to manage the GSM backend) and libgsm (an API to communicate with the GSM subsystem).
GPS: Similar to the GSM it includes GPS (a core GPS module) and GPSd (a daemon to manage the GPS backend).

The user interface kdrive: Represents the X server version for embedded platforms, it runs on a frame-buffer device and supports OpenGL.
matchbox: A window manager for embedded platforms.
GTK: The popular open source cross-platform widget toolkit. GTK is the base for the GNOME desktop environment.
ALSA: The Advanced Linux Sound Architecture (ALSA) provides the audio functionality.
Qtopia: A robust mobile phone and PDA platform from Trolltech.

The application framework Core (libmokocore): Provides the base functionalities of the framework: the switching device, reading/ writing data, application-to-application signaling, etc. It uses the Dbus and Gconf packets.
.Net (libmokonet): It makes use of libgsm, bluez-lib and libgps and provides basic networking functionality for the framework.
UI (libmokoui): It is responsible together with GTK and ALSA with providing the UI programming capabilities to the platform.
PIM (libmokopim): Together with the embedded Evolution data server it provides the core PIM (personal information management) capabilities.

3.2.2 The SHR Operating System

The Stable Hybrid Release OS, or SHR is the most currently used system by the OpenMoko community and the one running on the Neo FreeRunner smartphone used in this project. The OS is based on a Linux kernel and OpenEmbedded. A touch controlled desktop environment is provided by an X server environment and Illume2. A Freesmartphone.org (FSO) middleware is interfacing the GSM modem, GPS module and all other hardware integrated devices.

Interconnectivity With the help of USB Host support of the smartphone, it has become possible to communicate via the USB port with other devices. The installation of the FTDI driver allows the smartphone to also power up USB devices, this being the last step in achieving full USB Host capabilities for the FreeRunner.

3.2.3 Implementation on Android : Discussion

Since the Android OS for mobile platforms is, on top of being an open source project, one of the dominant player on the smartphone market at present, it would have been the natural choice for this experiment and in fact it represented one of the original ideas of the thesis. Even though the choice made for the project was for a different platform, the next discussion will address the USB host mode feature on Android devices. The purpose of this discussion is to illustrate how in the close future applications such as the network monitoring tool will have easier access to Android applications which are easier to develop and can reach a larger audience.

The Android SDK has made available, since the summer of 2011 the ”Android Open Accessory Development Kit” which included an API for USB. The API was made available for devices running Android 2.3.4 and later versions. Even though the name suggested otherwise, the API could only be used with an external USB host which in turn implied that a USB device could not be connected and, in addition to this, the device would not be powered by the Android platform. The first issue could be solved if gadget developers would create new hardware and firmware for their devices.

Some of these issues were more thoroughly resolved when the Android USB

Host API was released in the second half of 2011. It was only available for devices running Android 3.1 and later versions. The API allowed with a simple adapter that USB devices such as gadgets would be supported by Androids USB Host. There was one additional condition though, that the kernel used by the Android device supported the USB standard driver of the hardware - such as input, storage device, etc.

For those devices that would not be compatible (they don't have a standard driver) the solution for compatibility was the rooting of the mobile platform, and manual compilation of the kernel and the driver of the manufacturer. This operation was risky and not encouraged by hardware vendors.

As manufacturers grow more wary of the possibilities emerged from the interaction of USB devices with Android systems, or other mobile OSs as a matter a fact, comprehensible solutions are starting to be made available by vendors. It will probably not be long until Google will release an Android version which, backed up by a development kit will, allow the USB gadgets to be used at their full potential and give birth to a new generation of applications for smartphones.

Chapter 4

Application design and implementation

The two main components of the monitoring tool which represents the goal of this thesis project are the application needed for user interaction, analysis and interpretations of the results of the monitoring over WSNs and the mobile platform on which the application will run. In the Neo Freerunner two applications were created for this project: The main application which is extensively discussed in this chapter, and a small application which represents a script used for switching the smartphone from host mode into device mode and the other way around. The second application is called ”USB Mode”.

This chapter will describe design aspects of the application which was developed for this project and it is divided in three sections. Section 4.1 explains the requirements under which the application must run, section 4.2 defines the features that the application will offer and finally, section 4.3 presents the architecture of the application.

4.1 Requirements

The proposed project is a monitoring tool which allows the inspection of a part or a whole wireless sensor network deployment. This tool has to be installed on a mobile platform, more specific, a smartphone. The application must meet certain requirements in order to achieve its purpose.

Connectivity

A connection has to be established with a WSN which operates within a 802.15.4 network.

Portability

Portability is the first requirement of this tool as it needs to be able to detect WSN nodes in any environment. This requirement is dictated but the fact that the tool will facilitate the quick intervention for monitoring or inspecting remotely deployed WSNs which are malfunctioning or in need of maintenance.

Passivity

The passive character of the tool is required in order not to interfere with the functions of the network. In fact a correct formulation for this requirement would be: the presence of the tool should not influence the activity of nodes *unless this is specified by the user*. As it will be described in a subsection below, the application should be able to periodically inject packets into the WSN if it is set up in this way.

Range requirements

The coverage range should not be imposed to the entire deployment area of the WSN. Given the mobile nature of the system, the application should be able to hear the packets sent by a node and its neighbors. To study the behavior of nodes in a multi-hop path, the user of the application can follow the path using the tools other features, which will be presented below.

Logging

The application should be able to keep logs of the activity of the WSN.

Free localization

Since this application is not designed for a specific WSN deployment, it must use a method for localizing the nodes of the WSN, without using any additional hardware or information of the position of certain nodes. The accuracy of the localizing method should be within a range of 3 meters.

Packet receiving rate

The packet receiving rate should be specified. The acknowledgment of the data rate will help to avoid faulty interpretation of the data analysis. This incorrect interpretation would be made by using the application on a WSN characterized by a packet sending rate of its nodes which is too large for the capabilities of the sniffer to detect packets.

User intervention

User intervention should be achievable if it is decided that packets should be injected into the WSN for the correct evaluation of the behavior of the network.

4.2 Features

Built upon the requirements, the WSN monitoring tool will provide a number of features that will make its use practical.

- It will have a listening function, which will display on the screen of the smartphone the real time packet traffic from within the range of the sniffer.
- A replay function will also be available. It will allow setting breakpoints to pause the replay of the sniffed packets.
- A localization mode will be available to indicate with sufficient accuracy the physical location of a node.
- The application will offer the option to switch the channel frequency on which it listens for packets.
- The application will be equipped with a logic analyzer which will display information about the traffic of packets in the WSN.
- A protocol analyzer will be available to display and analyze the contents of the acquired packets.
- An injection function will be available so that at the indication of the user, user-defined packets will be periodically inserted into the network.
- Each inspection session will be logged to a location on the smartphone, for further analysis.

4.3 Architecture

The application architecture, displayed in Figure 4.1 contains several modules which interact with each other. The Main module is responsible for starting and stopping the application, logging and analyzing the data. The Localization module is used for the features related to finding nodes. The ReadSerial module is the component through which the the data is received from the TelosB mote and passed on to the Main module. The rest of the modules, Listen, Logic Analyzer, Protocol Analyzer and Injector have specific functions. A more detailed description of the software modules will follow in the section below.

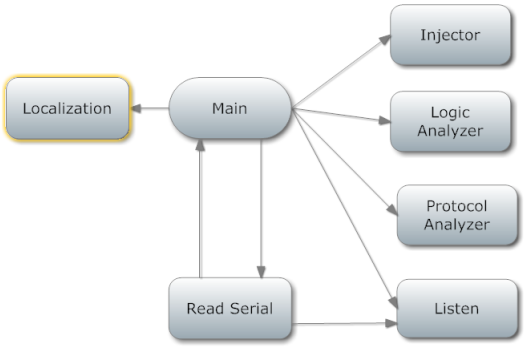

Figure 4.1: The application architecture

4.4 Software modules

The selected programming language for the development of the application was Python. The choice was made weighting a series of factors. First, the OpenMoko native toolkit was badly maintained. Secondly, SHR offers native Python support and last but not least the pygtk library, the Python version of the popular GTK+ library was available in the OpenMoko repository. Pygtk allows the development of comprehensible GUIs for applications.

The ReadSerial module

The ReadSerial module is used for handling the data which is received from the the serial port. The module is responsible for updating the data structures used by other modules, logging, and for analyzing the received data in order to detect packet losses. The data is at the same time sent to the Listen module which displays it in an on-line fashion, packet by packet, as shown in Figure 4.3.

The Main module

The main module is responsible for turning on or off the ReadSerial, Injector and Localizing modules. It also has the function of initializing all the other modules and the GUI.

Node localization

The node localization module is a module which can be enabled or disabled. Enabling it will disable the ReadSerial module, action which in turn will make the rest of the modules inactive. The reason why other modules are being paused during this activity is that it has been observed that localizing a node is a time consuming operation which involves movement. During this operation some of

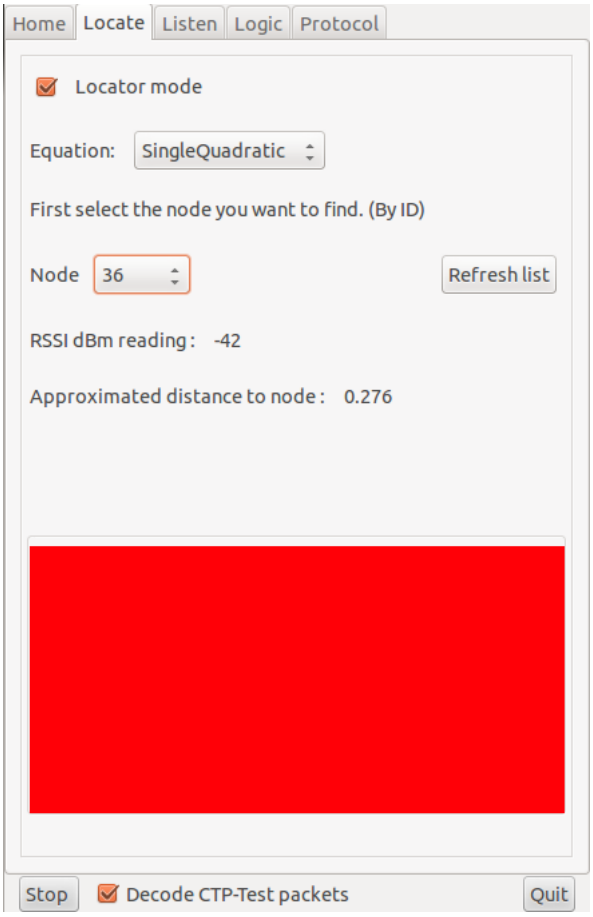

Figure 4.2: The node localization GUI

the nodes may fall out of range, the measured data being in this case inaccurate.

The localization is based on a formulas discussed in chapter 5. A choice is presented to the user for the preferred method of estimating the distance to nodes. A visual indicator will progressively change its color depending on how far the node is estimated to be. Four colors are used to indicate the position of the node: shades of red when the node is close, shades of purple when the node is up to 20m away and shades of blue for longer distances. If the sniffer mote gets out of relevant range, the color indicator will turn black.

Listen module

The Listen module provides a frame where raw packets are displayed as they arrive through the serial port. In addition to this, the dropped packets are also announced in the frame. Moreover, a ”Replay” function is available. During a replay, the received packets are displayed again, the user being given the possibility to

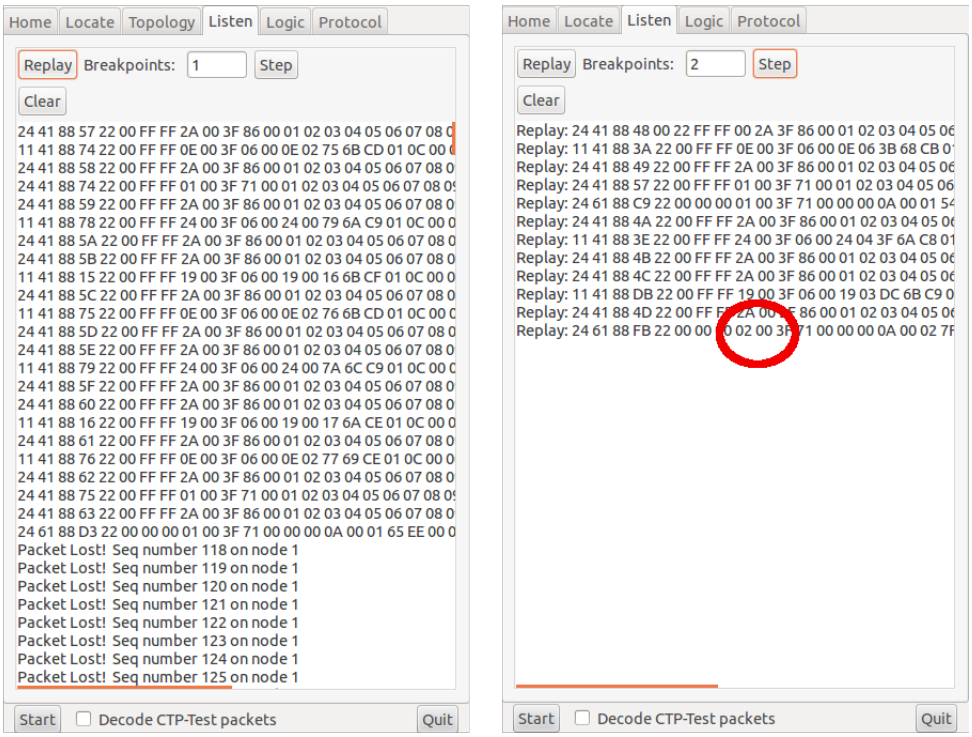

Figure 4.3: The Listen screen with the listen (left) and replay (right) functions

add one or more breakpoints.

Figure 4.3 demonstrates the two functions of the Listen module. In the right side, the red circle indicates the source address of the last displayed packet. The source address is in little endian format, which is why it appears as 0200. The display of packets has stopped at this packet because the breakpoint is set to the value ”2”. The left side of the figure illustrates the listening function of the module.

Packet analyzer

This module displays the arrival pattern of messages throughout the WSN. The packet analyzer is a tool which will be used to determine if nodes transmit in the right order, if there are any collisions, lost packets, or if a node is overloaded with packets. It also offers a summary of the last listening session.

In the logic analyzer each rectangle represents a time frame which is equal to the display rate parameter. A blue rectangle is the symbol for a sent packet while a red rectangle shows that the packet was sent after a packet drop was detected. A white rectangle shows that no packet has been sent in that time period.

The secondary function of this module is to provide a summary of information

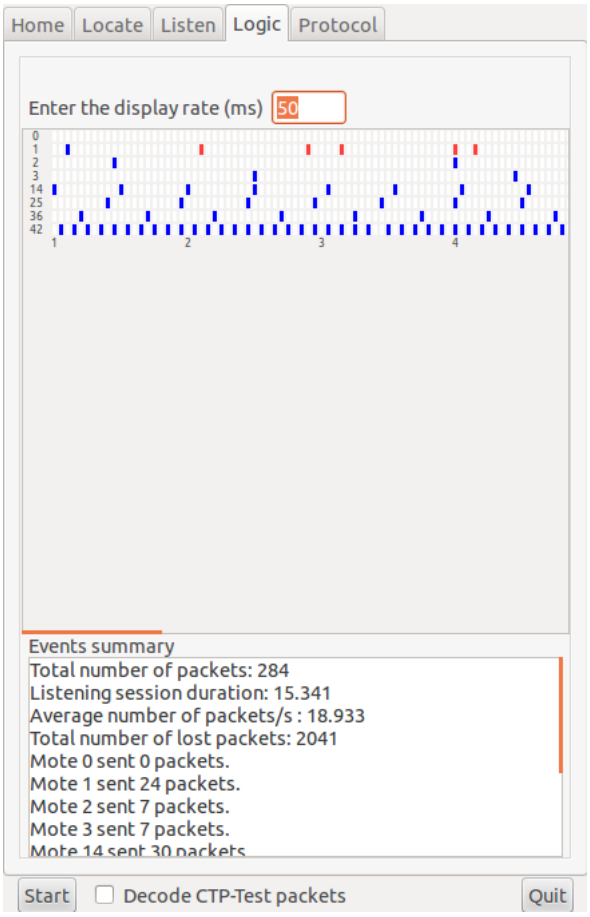

Figure 4.4: The logic analyzer and summary of the listening session

about the last listening session. The total number of packets and lost packets, the average amount of packets per second and the session duration in seconds are displayed on the first lines. The next lines offer information about each node, such as how many packets it has successfully sent or how many of its packets were lost.

Protocol decoding

This module decodes and displays the information from the received network packets. It uses the AM type field of ActiveMessage packets to classify the packets by protocol and decide how to display them. The MAC header data is also decoded. The possibility to browse through the packets from the last listening session is also given. Figure 4.5 shows a decoded CTP packet.

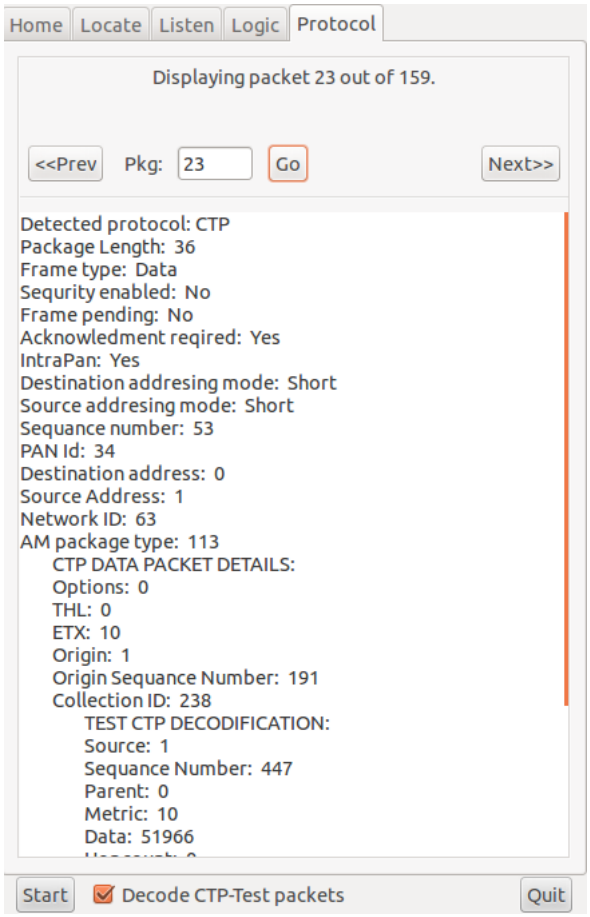

Figure 4.5: The protocol analyzer

Injector

Figure 4.6 shows the injector module, which is embedded in the home screen of the application.

The injector module is used for active inspection of the WSN. If the user decides so, he has the possibility to inject periodically messages into the WSN. The injector acts also as a node impersonator as it can send packets to the network with the source address field set to the value preferred by the user. In the same module, which also acts as a home screen, the user is given the choice to change the listening channel. The channels are numbered 11 to 26, the TinyOS standard. Default values for the data are set and can be changed by the user. The user can only enter integers lower than 255. If he does otherwise, a reminder of the allowed data range will be shown on the screen.

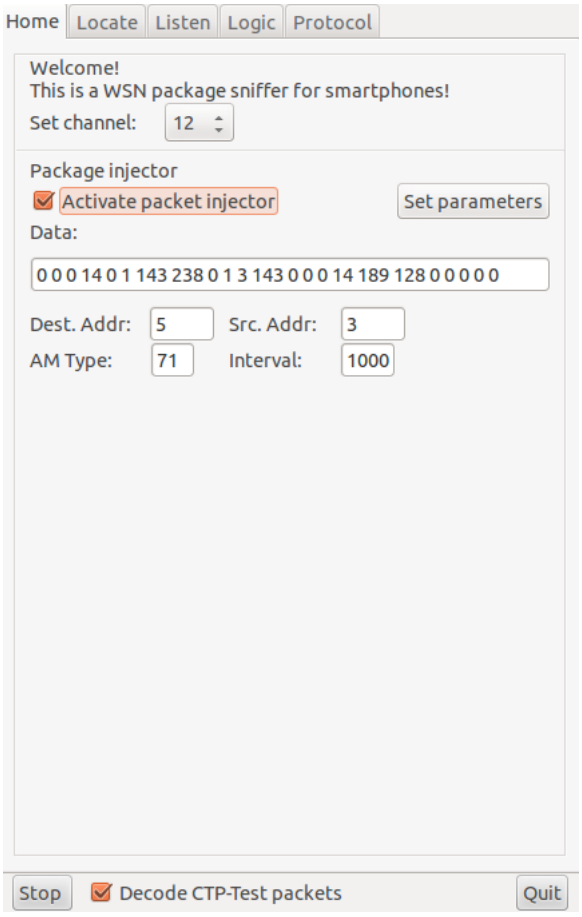

Figure 4.6: The packet injector

4.5 The TinyOS Program

On the mote side, a small TinyOS application was also created. The application is a modified version of the BaseStation15.4 application, which is distributed with TinyOS. Its main function is to act as a sniffer, which means it accepts all transmitted ActiveMessage packets and forwards them to the serial UART.

An additional feature for receiving packets through the UART was implemented. When a packet has arrived through the serial UART, it is processed and the mote may start or stop the periodic sending of a packet through the network or it may change the channel frequency on which it listens for packets.

Chapter 5

Node localization

In order for the previous described application to be able to point the location of nodes in an unknown WSN to an area of acceptable dimensions, a localization method that *uses no additional hardware* must be identified.

5.1 Localization method

Since a TelosB mote was chosen to act as a sniffer, the localization of the nodes in a monitored WSN will also be computed using the same device. Thus, as presented in section!2.3, the node localization method proposed in [17] will be reproduced.

In [17], the authors prove that under ideal conditions, a linear correlation can be achieved between the RSSI value of the packets received from a TelosB mote and the logarithm of the distance between the target mote and the ”seeker” mote.

The choice for this localization method was made because, in the case of both choices of the present thesis and the authors of [17] choice, the chosen mote for the experiment was a TelosB mote.In addition to this, they have proved the concept that the distance can be measured (in ideal conditions) by correlating it with the Radio Signal Strength Indicator (RSSI) value of received packets, without using additional hardware.

5.2 Localization model

The RSSI is a metric which measures the signal strength at the receiving end of a packet. The value, which ranges usually between -45dBm and 100dBm, is measured in dBm to indicate the signal power. The required energy power threshold of the a node for receiving packets determines the lower value while the high value indicates the maximum signal strength. Figure 5.1 gives a visual indication of the sending range of a mote correlated to the RSSI value.

The Crossbow TelosB mote used for this project uses a CC2420 2.4GHz RF

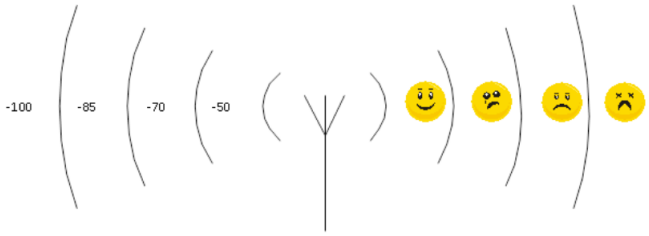

Figure 5.1: The RSSI coverage

transceiver. CC2420 is equipped with a RSSI which returns a value that can be read as the 8-bit complement of the signed 2 of the $RSSI.RSSI\_VAL$ register.

Localization Model As in [17], the received signal strength at the RF pins has the following expression :

$$RSS = RSSI\_VAL + RSSI\_OFFSET[dBm] \tag{5.1}$$

The $RSSI\_OFFSET$ has empirically been found to be approximately -45([2]), which gives the Received Signal Strength value expression of :

$$RSS = RSSI\_VAL - 45[dBm] \tag{5.2}$$

The correlation between the RSS value and distance is given by:

$$RSS = 10log(P/P_{ref}) \tag{5.3}$$

Where P is the received signal power and P_{ref} is the reference power. Thus:

$$RSS\ \alpha\ log(p) \tag{5.4}$$

The received signal power is indirectly proportional with distance:

$$P\ \alpha\ 1/D^n \tag{5.5}$$

Where D is the distance from the sniffer node to the targeted node and ”n” is the path loss exponent factor. From the equations 5.3, 5.4 and 5.5 :

$$RSS\ \alpha\ 10log(1/D^n) \tag{5.6}$$

$$RSS = -10nlog(D) + C \tag{5.7}$$

Equation 5.7 demonstrates the linear correlation between the logarithm of the distance and the RSS value. In this equation C is a constant. To refine the 5.7 equation:

$$RSS = -mlog(D) + C \tag{5.8}$$

Where m is the slope parameter of the linear equation 5.7. m is derived from the path loss exponent factor which has the expression

$$n = m/10 \tag{5.9}$$

Also from 5.8 the expression of the estimated distance is:

$$D = 10^{-(\frac{RSS-C}{m})} \tag{5.10}$$

Equation 5.10 represents the mathematical expression of the estimated distance in correlation with the RSSI value. In section 6.1 of the next chapters the evaluation of this formula will be discussed.

Chapter 6

Evaluation

6.1 RSSI Localization

In order to confirm the theoretical analysis from chapter 5 which concluded that there is a linear correlation between the RSSI value and the log_{10} of the distance between two nodes, an experimental validation was pursued. RSS measurements are very sensitive to noise and interference which produce incorrect values which, in turn, lead to interpretation errors. Because of this the experiment has been carried out in an open-space environment (a park) which was free from interference generated by any device using the range of 2.4GHz.

The experiments involved a stationary TelosB node. The mote was connected to a USB power supply in order to keep the power level constant throughout all experiments. In addition to this, the RF power level of the mote was set as the value of the default level. The measurements were made in two phases: the calibration phase, and the evaluation phase.

The calibration measurements were taken using the built application on the Neo FreeRunner smartphone running in Localization Mode, which was kept at a constant height of 110 centimeters. The measurement data was collected on a distance of 30 meters, samples being collected at every carefully measured meter.

Taking in consideration that the antenna of TelosB motes is not isotropic, additional precautions were needed for a correct collection of the values. More specifically, the orientation of the sniffer mote was always the same during the experiment while the orientation of the targeted stationary mote was varied with each set of measurements. The measurements were repeated for four positions of the mote, each of them being characterized by a 90 degree orientation difference from the previous one.

The collected data acquired from the measurements made on each orientation was averaged and it was used generate the plot showed in Figure 6.1. The RSSI reading can be seen along the y axis while the $log_{10}(Distance)$ is represented

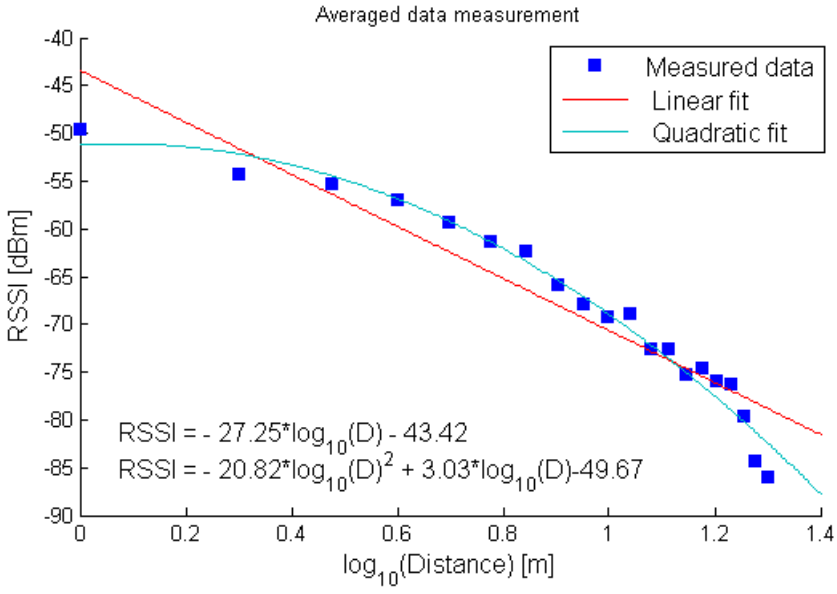

Figure 6.1: The general RSSI dependency on the log_{10} function of distance

on the x axis.

The data was fitted with a curve which in resulted in the linear coefficients for the 5.8 equation as follows:

$$m = -27.25$$

$$C = -43.42$$

$$n = 2.725$$

This turns the expression of the equation 5.8 into:

$$RSS = -22.592log(D) - 42.594 \tag{6.1}$$

Or to give the expression of the estimated distance in relationship with the RSSI reading from equation 5.10 we have:

$$D = 10^{-\left(\frac{RSS+42.594}{22.592}\right)} \tag{6.2}$$

In the same time it was observed that a quadratic fit would actually be more suitable for this set of data. As it can be seen, the quadratic fit seems to be a better choice than the linear equation, even if it is not backed up by any theoretical analysis. Because of this aspect, the decision to implement in the localization mode a choice of the method of calculation for the distance was made. The possibility to choose the algorithm will facilitate the comparison between the two methods as seen below.

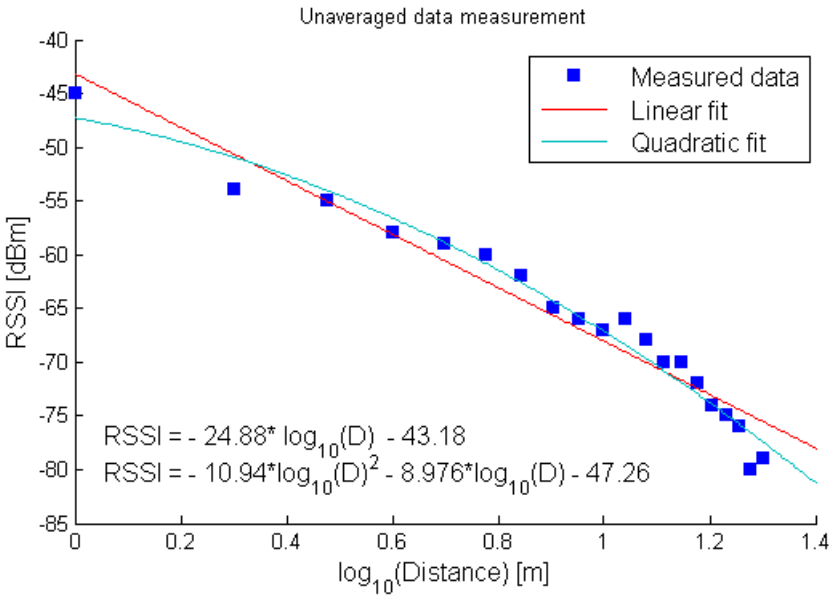

Figure 6.2: The orientation-specific RSSI dependency on the log_{10} function of distance

The log_{10} of the distance in this case is the most suitable out of the two solutions of the equation:

$$RSSI = -20.82 * log_{10}(D)^2 + 3.03 * log10(D) - 49.67 \tag{6.3}$$

As mentioned before, the measurements were made across a straight line at points situated one meter from each other, for a total distance of 30 metes. Despite the fact that 30 meters were measured, only the data obtained from the last 20 meters was taken in consideration. This happened because the values which were read for the last 10 meters of measurements presented enormous inconsistencies. In addition to this, at a careful study, it could be seen that the measured values corresponding to a specific orientation (when the USB of the mote is facing the sniffer mote) showed a significant difference in values from the ones with different orientations. Due to this fact, the decision was made to also plot the ”direct orientation” data separately, and include its equations in the choice of distance calculation methods. The sole purpose of this decision was to be able to better compare the real-life application of these methods. Figure 6.2 shows the plotted data.

In this case the linear coefficients needed for calculating the distance are.

$$m = -24.88$$

$$C = -43.18$$

$$n = 2.488$$

Thus, equation 5.10 becomes:

$$D_{specific} = 10^{-(\frac{RSS+43.18}{24.88})} \tag{6.4}$$

While the quadratic expressions which will be used for calculating the distance is :

$$RSSI = -10.94 * log_{10}(D_{specific})^2 - 8.976 * log10(D_{specific}) - 47.26 \tag{6.5}$$

In the experimental evaluation six types of measurements were made in order to asses the distance estimation formulas mentioned before. The data was collected in the same manner, in a wide open space, free of interference. The ”lost” mote was placed on the ground with the antenna facing upwards and it was powered by a USB power supply. The measurements were made with the smartphone running the sniffer application and the sniffer mote was held at a height of 110 centimeters of the ground. The sniffer orientation of the mote was constant, to be more exact the USB port of the mote was facing the smartphone.

In the first phase of the evaluation, all the four methods of determining the distance were used to find a mote which was had *ideal orientation* (the USB port is facing the sniffer mote). The methods given names correspond to the above equations:

- Orientation Independent Linear method - equation 6.2
- Orientation Independent Quadratic method - equation 6.3
- Single Orientation Linear method - equation 6.4
- Single Orientation Quadratic method - equation 6.5

The quadratic method has a small advantage in accuracy over the linear method, especially for distances larger than 5 meters, while the linear method performs better for the close range estimation. Overall, both methods prove to be quite reliable, considering they both can indicate the location of the mote within a 3 meter range. To be more precise, the average error of distance over 20 meters was less than one meter.

The formulas for the specific orientation had an average error of the estimations of distances of around 0.7 meters, in the case of the linear estimation method, and only 0.56 meters using the quadratic estimation method. This results validates the model achieved in Chapter 5 and also proves that a better estimation can be achieved using a different theoretical approach. In this case the better formula was found only empirical.

The last evaluation of the distance estimation methods was made simulating a closer to real conditions example. A mote was placed on the ground with random orientations. Measurements were performed at distances situated one meter from each other on a distance of 20 meters. Unlike the previous tests, the real

Number of p/sec	ms	Packets lost for	Application behavior
22	51	0	Normal
54	20	0	Normal
68	15	0	Normal
100	10	0	Normal
113	9	0	Normal
128	8	0	Normal
148	7	0	Unstable

Table 6.1: Evaluation of the sniffer capacity when packets with payloads of 1 bytes are sent

conditioned test did not offer the same accurate results even by using the formulas which were computed using measurements taken on multiple orientations of the ”lost” motes. Moreover, for distances which were larger than 8 meters it was observed that the readings were inconsistent and off the scale. For these distances, the average error generated by the formulas is still kept within the requirement of 3 meters.

6.2 Sniffer capacity

A very important metric for the sniffer application is the capacity of receiving packets of the sniffer application. In order to asses this metric, an experimental setup composed of a sending mote and the smartphone with the attached sniffer mote was used.

On the sniffer end, the presence of lost packets was used as an indicator that the capacity of the sniffer mote was exceeded. Packets are lost when the receiver buffer overflows. The CC2420 datasheet [2] reveals more information on the receiver buffer. A FIFO stack of 128bytes is used for absorbing the receiving packets. Two kind of packets were used for this evaluation : one type which had a 1 byte payload and another type which had 23 bytes of payload, simulating a CTP packet. MAC and physical layer add headers and footers of a total of 18 bytes to the payload [3], which brings to the final length of the evaluation packets to be 19 and 41 bytes. This suggests that the receiver FIFO stack can accommodate 6 short evaluation packets or 2 CTP packets. The authors of [10] have measured the processing time of a packet in the receiver FIFO and found it to be close to 4.5 ms. By this result it is suggested that the sniffer should be able to handle more than 200 packets per second.

One more step was needed in order for the tests to be conclusive: the sending mote had to be able to send packets continuously, without any backoffs caused by the MAC protocol. The Carrier Sense Multiple Access, or CSMA [27], needed

Number of p/sec	ms	Packets lost for	Application behavior
22	51	0	Normal
54	20	0	Normal
68	15	0	Normal
100	10	0	Normal
113	9	0	Unstable

Table 6.2: Evaluation of the sniffer capacity when packets with payloads of 23 bytes are sent

to be turned off to assure the contentiousness of the packet sending action of the sending mote. To be more specific, in the TinyOS code, using the RadioBackOff interface provided by the ActiveMessage component, the initial and congestion backoff values were changed to 0 and the Clear Channel Assesment, or CCA, request was set to false.

For the evaluation of the packet receiving capacity of the sniffer application, the testing scenario involved decreasing the packet sending interval of the sending mote from a value of 50 ms down to a rate of 5 ms. As it can be seen in the Tables 6.1 and 6.2, the tests stopped at a value of 7 or 8 ms, depending on which size of the packets were used in the experiment.

As it can be seen in both Table 6.1 and 6.2, there were no lost packets during the experiment. However, the experiment could not continue for sending rates of over 130 packets per minute because of a design issue with the application which proves to be unstable when high packet rates are delivered to it. This issues has to do with the fact that at high incoming packet rates, the messages which are sent through the serial UART between the application and the sniffer mote are corrupted.

In these conditions, it can be concluded that even if the capacity of the sniffer mote can theoretically reach up to 250 packets handled per minute, the correct workload for the sniffer application with CTP-sized packets is 113 packets per minute, while with short packets it lies at 128 packets per minute.

6.3 Sniffer capacity with enabled packet injection

To make a further step in the evaluation of the sniffer application, the sniffer capacity of receiving packets was evaluated in the scenario when the injector function of the mote is active. Tests were performed in a similar way as the tests described in the previous section. Having a mote sending packets at fixed rates, which can be considered to be the background traffic from the WSN, the packet injection period of the sniffer mote was decreased from 2 seconds down to 25 milliseconds. The

Injector rate [s]	Packets lost	Packets Total	Loss %	Status
2	0	3300	0.00%	Normal
1.5	2	4264	0.05%	Normal
1	5	4250	0.12%	Normal
0.5	4	2291	0.17%	Normal
0.25	24	3036	0.79%	Normal
0.2	6	2206	0.27%	Normal
0.1	41	3532	1.16%	Normal
0.05	125	3360	3.72%	Normal
0.025	n/a	n/a	n/a	Unstable

Table 6.3: Evaluation of the sniffer capacity with injection with traffic of 85 packets/s

sent packets had a fixed 23 byte payload.

During the tests, packet losses were counted in order to evaluate the sniffers performance under specific workloads of both receiving and transmitting packets. Table 6.3 represents the most conclusive test in this direction as it displays the highest rate of incoming packets under which the application performs normally. It can be observed that on the last row, the application stopped responding and no data is available from the setup involving an incoming data rate of 85 packets per second and a transmission rate of 40 packets per second. Other test results are available in the Tables 8.1, 8.2 and 8.3 of the Appendix. A small number of tests were possible even with an incoming packet rate of 100 packets per second, however, as it can be seen in Table 8.3 the tests started to induce an unstable behavior of the application even for injection rates of 2 packets per second.

Turning back the focus on the lost packets, it can be seen in Table 6.3 that packet losses do occur when the injector is active. The question of setting a threshold to how many packets can be lost without having the loss of data affect the inspection operation does not make the point of this study. From the results of the aforementioned tables, it can only be concluded that for incoming packet rates of up to 85 packets per second and outgoing packet rates of up to 20 packets per second, not more than 5% of packets are ever lost.

6.4 Packet loss detection

The last metric which was evaluated is the packet loss detection accuracy. For this evaluation disturbing traffic was generated from a mote, which will be called the ”disturbing mote”. The disturbing mote sends periodical packets in the purpose of measuring the packet losses from another mote, which will be called ”the observed mote”. The observed mote also sending packets at a lower rate. The disturbing traffic generation was made as in the previous experiments with no CSMA

CTP packets	Disturbing Traffic Sending Interval [ms]	Number of lost packets	Estimation of lost packets number
1411	1000	0	1.53
1161	500	0	2.52
1937	250	6	8.41
1290	200	7	7.00
1231	100	10	13.32
1324	50	31	28.50

Table 6.4: The collision detection test results

and by varying the transmission rate of 23 byte CTP simulated packet transmission from 1 to 20 per second. The second sending mote was also sending a CTP packet every second.

To estimate the losses of the observed mote, the number of collisions between the packets of the observed and the ones of the disturbing mote was calculated. The number of collisions can be estimated by first acknowledging that the disturbing traffic generation follows a Poisson distribution with the arrival rate λ, equal to the packet sending rate of the disturbing mote. Then the probability that a packet from the disturbing mote is sent during the interval t is $1 - e^{\lambda * t}$ ([35]). Considering that t is the interval when the observed mote is sending a packet then the previous formula denotes the probability of collision between the packets sent by the two motes. Following up, considering that the observed mote has sent N packets, then the estimated number of collisions between the packets of the observed mote and the packets of the disturbing mote is

$$Collisions = N * (1 - e^{\lambda * t}) \tag{6.6}$$

The only unknown item from the 6.6 equation is the t, the transmission time of a packet from the observed mote. The payload of the packets sent by the observed mote is 23 bytes, the PHY level adds 5 more ([3]) and the CC2420 message_t structure adds 13 more bytes (11 for the header and 2 for the footer of a message_t packet). In total a message has 41 bytes or 328 bits. The CC2420 RF can transmit with up to 250kbps. Therefore

$$t = \frac{328}{250000} = 0.001312[s]$$

Table 6.4 shows the number of packets lost for different packet sending rates of the disturbing mote and the estimations provided by the 6.6 equation. It can be observed that the differences between the estimation and the real number of lost packets are rather small which proves that the losses are as anticipated caused by packet collisions

Packets/s	CPU Load	Memory Usage
1	15	13.1
2	15.8	13.2
10	16.4	13.3
20	17	13.4
50	20.6	13.3
100	25.3	13.8
130	33.2	14.3

Table 6.5: Performance test - the network activity scenario

6.5 CPU load and Memory Usage

An evaluation was conducted on the impact of the application on the performance on the FreeRunner smartphone. The CPU load and Memory usage were measured in three different scenarios. Due to compatibility issues the profiling of the application was not made with an off-the-shelf software but with a self-developed script which only outputs the memory usage and CPU load of a process.

As stated in Chapter 3 the Neo FreeRunner is equipped with a 400MHz ARM processor and 128MB of SDRAM.

For the first test scenario the application was idle. The idle application uses 13% of the available memory and loads the CPU by 2.2%. The rather large usage of memory can be attributed to the GUI The application does not use large data structures but on the other hand it uses many GTK objects.

The next test evaluated the impact of the application which is used in Localizing mode over the general performance of the smartphone. Tests were conducted by turning the localizing mode on while the WSN was configured to have traffic rates from 5 to 128 packets per second. Throughout these tests, the CPU load was of 5.6% while the memory usage did not exceed 13.1%.

The last conducted test was the performance impact of the application when listening is performed. The traffic rate of the WSN was varied between 1 and 128 packets per second, which is also the best case scenario for the sniffer capacity of hearing packets without losses. The results, displayed in Table 6.5 show clearly that the CPU load increases constantly in correlation with the traffic rate of the WSN. This is justified by the fact that the at higher traffic rates, the processor has to handle more requests in the same interval of time. The memory usage has a much softer increase in value, which, because the application does not use large structures of data, is caused by the increase of data which needs to be displayed in the GUI.

Chapter 7

Discussions

7.1 Project challenges

This project represented a great opportunity to actively learn about wireless sensor networks while working every day with issues met in this field. One of the things I appreciate most was that I managed at the end of the project to build my own TinyOS applications for the tests. In the same time, forced by the lack of support for the native toolkits of the OpenMoko platforms, I gained experience with Python, a language which I have never used before.

One of the biggest challenges of the project was the finding a localization technique. Much research has been done in this direction but so far it seems that without any additional information about the locations of some of the nodes, or without additional hardware, accurate localization can not be achieved in non ideal conditions.

In the early phases of the project, the OpenMoko Neo FreeRunner seemed to be a good choice of hardware for the implementation of the application. However, due to badly maintained repositories and constant bugs of many popular Linux packages, using the Neo FreeRunner for implementation has turned into the most challenging part of the project.

One of the issues caused by the previously mentioned issues, building the GUI in the pygtk library, in the scenario where the pygtk library supported by the OpenMoko distribution is 2.10.4, a very old release. This library offers little support for dynamic elements and even if sometimes these problems can be overcame by using threading (as in the Listen module for the listening function), sometimes the only solution is for large structures of data to be processed all at once. This of course causes delays. Such a delay can be observed when the stop button of the application is pressed and the the logic analyzer table is computed. The implementation of the logic analyzer was the most problematic and at the end it was solved by using a table of images (white, blue and red). This table can not be refreshed, so this is why after each listening session it gets destroyed and recom-

puted.

Finally, benchmarking the application for the CPU load and memory usage was proven to be a very unexpected issue. As the OpenMoko OS has several known issues with the native toolkit, a profiler could not be installed with the purpose of evaluating the performance. Because of this a self made python script was used to sample the CPU and memory parameters every half of second.

7.2 From prototype to product

If the possibility of turn this thesis project into a commercial application would show itself, the first thing that needs to be done is to port the application to the Android OS. Android smartphones are very popular and the trend seems only to be increasing. As mentioned before, Google has already released SDKs with USB support, but at the moment the main issue with programming such an application on Android is that most USB devices are not supported by the kernel of the Android OS. Probably it is just a matter of time until Google will offer generic drivers for more USB devices.

The application would also definitely benefit from being tested and used on recent generations of smartphones which have a much better specifications.

By running on an Android device, the GUI of the application could also be developed in a different language, and make use of animation and dynamic displays of information. The application would also have access to the other equipment of the smartphone. For example it could access a camera. In this way, by integrating augmented reality, a new method for node localization can be found which would make the finding of the nodes become a more trivial task.

Chapter 8

Conclusion

8.1 Conclusion

The number of wireless sensor networks is constantly growing as they can be used in a large variety of fields ranging from environmental monitoring, industrial measurements, military applications and so on. The WSNs can sometimes be deployed in inaccessible areas or areas which pose a high degree of risks for humans due to the profile of the terrain, wildlife, or environmental conditions. For many of these deployments, especially for the ones which do not benefit from a GPRS or Internet link, the monitoring and inspecting operations of the WSN prove to be a difficult task. The difficulty arises mainly from the fact that laboratory equipment needs to be brought along and connected to the WSN. The goal of this thesis was to design, implement and evaluate an application for smartphones which can be used for monitoring and inspecting WSNs.

In this purpose two major issues have risen. The first one was the achievement of the interconnection between a smartphone and a WSN, as the two domains using totally different communication standards. The second one was to identify a localization method that will allow the finding of nodes without any information about the position of other nodes or help from any additional hardware.

The first issue weighted heavy in the matter of the choosing the smartphone platform. An OpenMoko Neo Freerunner smartphone running a SHR Linux based OS was chosen for the implementation of the project. The Neo Freerunner offers full USB Host support and thus allowed the interfacing with WSNs through an TelosB sensor node which was connected through the USB port. The TelosB acted as a packet sniffer, a packet injector but also as a serial forwarder for the smartphone.

The localization of nodes was made by correlating the distance from the sniffer to a node with the Received Signal Strength value of the packets which were received from the target. The analytical equation showed a linear dependency between the two parameters. However, from the data gathered from measurements, besides the linear dependency, an accurate quadratic fit was also observed. Both

methods were evaluated empirically and they produced results which leave room for interpretations. As the RSSI spectrum of 802.15.4 antennas is not isotropic, in the case of specific orientations of the node the estimation formulas were very accurate, while in other cases their accuracy dropped dramatically. Finally, it was demonstrated that for short distances, nodes can be localized to an area of within 3 meters using the RSSI value, regardless their orientation.

The application itself was built using the Python language. It provides a GUI for the inspection tool and offers an attractive set of features, some of which can be very useful for debugging faults that appear in the WSN. Users have the possibility of seeing the packets that are sent over the WSN displayed on the screen of their smartphones. They can replay the last recorded sequence of packets, pausing at preselected breakpoints. After each listening session, the packets are stored in the memory of the smartphone from where they can be transferred to a workstation. At the same time, the packets are analyzed and a logic map of the packet arrivals is displayed. Information about the listening session such as number of packets, of packet losses, average rate of packets, individual node sending and failure notices is available after each session. The last passive inspection feature is the Protocol Analyzer which decodes the payload of each packet and displays it to the user. Active inspection is also possible, the application being able to command the attached the TelosB mote to ”impersonate” another node and send packets into the WSN at specific intervals, with a specific destination.

The testing and evaluation of this project was proven to be difficult because, being a monitoring tool, the application manipulates a lot of data but takes few inputs and generates few outputs that can be tested with classic methods. However, for evaluation purposes the capacity of the TelosB mote for processing packets has been determined both in the cases of active inspection and monitoring of a WSN. For the passive monitoring, the sniffer capacity is between 100 and 130 packets per second, depending on the message payload. For active monitoring, the sniffer capacity is of 85 packets per second. Another validation for this tool was made for the accuracy of the dropped packets reports. The number of lost packets was estimated and also empirically measured, the two values being relatively close to each other. In the last evaluation test, the performance impact of the application over the Neo FreeRunner smartphone was performed. At high traffic rates, the application can generate large CPU loads (up to 33%) which affect the overall perfomance of the phone.

The project has been successfully implemented, it offers an attractive set of features, it is portable and, hopefully, it will one day ease the tasks and the laboratory equipment carry loads of technicians and researchers who needs to inspect WSN deployed in remote hard to reach locations.

8.2 Future Work

The goal of this thesis has been met and an application was designed with the purpose of inspecting hard to reach deployments of WSNs. Even if most functions have been implemented already, a few ideas of further development have been identified:

- Porting the system to the popular Android and iOS platforms. These operating systems are running on the majority of smartphones on the market. Since predictions say that soon these OSs will also benefit from USB Host support, porting the implemented tool to these platforms seems to be the logical next step.
- By changing the platform to a more recent one, a more dynamic GUI can be implemented. The Logic Analyzer and the Protocol Analyzer would then be able to be updated on-line during the listening session.
- A better localization algorithm would render better results in finding lost nodes. An improvement is needed here, especially for the indoor environments.
- Last, but not least a topology detection and display feature can be implemented as an extra module for the current application.

Bibliography

[1] Telosb datasheet: http://www.willow.co.uk/telosb_datasheet.pdf , crossbow.

[2] Chipcon CC2420 Datasheet: http://focus.ti.com/lit/ds/symlink/cc2420.pdf, Texas Instruments, 2007.

[3] Ieee standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements. part 15.3: Wireless medium access control (mac) and physical layer (phy) specifications for high rate wireless personal area networks (wpans) amendment 2: Millimeter-wave-based alternative physical layer extension. *IEEE Std 802.15.3c-2009 (Amendment to IEEE Std 802.15.3-2003)*, pages c1 –187, 12 2009.

[4] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102–114, 2002.

[5] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A line in the sand: a wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 46(5):605 – 634, 2004. ¡ce:title¿Military Communications Systems and Technologies¡/ce:title¿.

[6] Xirong Bao, Fupeng Bao, Shi Zhang, and Lei Liu. An improved dv-hop localization algorithm for wireless sensor networks. In *Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on*, pages 1 –4, sept. 2010.

[7] Jan Beutel, Kay Römer, Matthias Ringwald, and Matthias Woehrle. Deployment techniques for wireless sensor networks. In G. Ferrari, editor, *Sensor Networks: Where Theory Meets Practice*, Heidelberg, 2009. Springer.

[8] P. Brida, J. Machaj, J. Benikovsky, and J. Duha. A new complex angle of arrival location method for ad hoc networks. In *Positioning Navigation and Communication (WPNC), 2010 7th Workshop on*, pages 284 –290, march 2010.

[9] Wu Chengdong, Chen Shifeng, Zhang Yunzhou, Cheng Long, and Wu Hao. A rssi-based probabilistic distribution localization algorithm for wireless sensor network. In *Information Technology and Artificial Intelligence Conference (ITAIC), 2011 6th IEEE Joint International*, volume 1, pages 333 –337, aug. 2011.

[10] J. Cote, Bing Wang, Wei Zeng, and Zhijie Shi. Capability and fidelity of mote-class wireless sniffers. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1 –6, dec. 2010.

[11] Meijuan Gao, Fan Zhang, and Jingwen Tian. Design and implementation of wireless sensor network data collection terminal based on arm9. In *Computing, Communication, Control, and Management, 2008. CCCM '08. ISECS International Colloquium on*, volume 2, pages 587 –590, aug. 2008.

[12] Sinan Gezici. A survey on wireless position estimation. *Wireless Personal Communications*, 44:263–282, 2008.

[13] J.A. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile. Ieee 802.15.4: a developing standard for low-power low-cost wireless personal area networks. *Network, IEEE*, 15(5):12 –19, sept.-oct. 2001.

[14] Wei-Wei Ji and Zhong Liu. An improvement of dv-hop algorithm in wireless sensor networks. In *Wireless Communications, Networking and Mobile Computing, 2006. WiCOM 2006.International Conference on*, pages 1 –4, sept. 2006.

[15] Matthew Keally, Gang Zhou, Guoliang Xing, Jianxin Wu, and Andrew Pyles. Pbn: towards practical activity recognition using smartphone-based body sensor networks. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 246–259, New York, NY, USA, 2011. ACM.

[16] Xin Kuang and Jianhua Shen. Snds: A distributed monitoring and protocol analysis system for wireless sensor network. In *Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010 Second International Conference on*, volume 2, pages 422 –425, april 2010.

[17] P. Kumar, L. Reddy, and S. Varma. Distance measurement and error estimation scheme for rssi based localization in wireless sensor networks. In *Wireless Communication and Sensor Networks (WCSN), 2009 Fifth IEEE Conference on*, pages 1 –4, dec. 2009.

[18] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp., april 2006.

[19] Youngki Lee, Younghyun Ju, Chulhong Min, Jihyun Yu, and Junehwa Song. Mobicon: Mobile context monitoring platform: Incorporating context-awareness to smartphone-centric personal sensor networks. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012 9th Annual IEEE Communications Society Conference on*, pages 109 –111, june 2012.

[20] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 88–97, New York, NY, USA, 2002. ACM.

[21] Guoqiang Mao, Bar Fidan, and Brian D.O. Anderson. Wireless sensor network localization techniques. *Computer Networks*, 51(10):2529 – 2553, 2007.

[22] J. Mung and J. Yen. Fundamental limits and simulations on time difference of arrival source localization using ultrasound signals. In *Ultrasonics Symposium (IUS), 2010 IEEE*, pages 1791 –1794, oct. 2010.

[23] D. Niculescu and B. Nath. Dv based positioning in ad hoc networks. *Telecommunication Systems*, 22(1):267–280, 2003.

[24] Paritosh Padhy, Kirk Martinez, Alistair Riddoch, H. L . Royan Ong, and Jane K. Hart. Glacial environment monitoring using sensor networks. In *Real-World Wireless Sensor Networks*, 2005. Event Dates: June 20-21 2005.

[25] Jeongyeup Paek, K. Chintalapudi, R. Govindan, J. Caffrey, and S. Masri. A wireless sensor network for structural health monitoring: Performance and experience. In *Embedded Networked Sensors, 2005. EmNetS-II. The Second IEEE Workshop on*, pages 1 – 10, may 2005.

[26] A. Panwar and S.A. Kumar. Localization schemes in wireless sensor networks. In *Advanced Computing Communication Technologies (ACCT), 2012 Second International Conference on*, pages 443 –449, jan. 2012.

[27] I. Ramachandran and S. Roy. Wlc46-2: On the impact of clear channel assessment on mac performance. In *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pages 1 –5, 27 2006-dec. 1 2006.

[28] B.C. Rolando, B.O. Julio, and P.A. Esaias. Controlling digital dimmer through mobile phone. In *Electronics, Communications and Computer (CONIELECOMP), 2010 20th International Conference on*, pages 57 –61, feb. 2010.

[29] M. Saxena, P. Gupta, and B.N. Jain. Experimental analysis of rssi-based location estimation in wireless sensor networks. In *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, pages 503 –510, jan. 2008.

[30] L. Selavo, A. Wood, Q. Cao, T. Sookoor, H. Liu, A. Srinivasan, Y. Wu, W. Kang, J. Stankovic, D. Young, and J. Porter. Luster: wireless sensor network for environmental research. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 103–116, New York, NY, USA, 2007. ACM.

[31] S. Singhal, A.K. Gankotiya, S. Agarwal, and T. Verma. An investigation of wireless sensor network: A distributed approach in smart environment. In *Advanced Computing Communication Technologies (ACCT), 2012 Second International Conference on*, pages 522 –529, jan. 2012.

[32] I. Stoianov, L. Nachman, S. Madden, T. Tokmouline, and M. Csail. Pipenet: A wireless sensor network for pipeline monitoring. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 264 –273, april 2007.

[33] Yong Sun, Bo Jing, and Zonglin Zhang. Location discovery and error analysis of wireless sensor networks based on difference of arrival time of beacon signals. In *Information Acquisition, 2007. ICIA '07. International Conference on*, pages 86 –89, july 2007.

[34] D. Tholl and M. Fattouche. Angle of arrival analysis of the indoor radio propagation channel. In *Universal Personal Communications, 1993. Personal Communications: Gateway to the 21st Century. Conference Record., 2nd International Conference on*, volume 1, pages 79 –83 vol.1, oct 1993.

[35] P. Van Mieghem. *Performance Analysis of Communications Networks and Systems*. Cambridge University Press, 2006.

[36] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 381–396, Berkeley, CA, USA, 2006. USENIX Association.

[37] Lingfei Wu, M.Q.-H. Meng, and Huawei Liang. A beacon selected localization algorithm for ad-hoc networks of sensors. In *Mechatronics and Automation, 2009. ICMA 2009. International Conference on*, pages 3091 –3096, aug. 2009.

[38] Enyang Xu, Zhi Ding, and S. Dasgupta. Source localization in wireless sensor networks from signal time-of-arrival measurements. *Signal Processing, IEEE Transactions on*, 59(6):2887 –2897, june 2011.

[39] Yu Yang, Peng Xia, Liang Huang, Quan Zhou, Yongjun Xu, and Xiaowei Li. Snamp: A multi-sniffer and multi-view visualization platform for wireless sensor networks. In *Industrial Electronics and Applications, 2006 1ST IEEE Conference on*, pages 1 –4, may 2006.

[40] Won-Jae Yi, Weidi Jia, and J. Saniie. Mobile sensor data collector using android smartphone. In *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*, pages 956 –959, aug. 2012.

[41] Dengyi Zhang, Feng Liu, Lei Wang, and Yuanxiu Xing. Dv-hop localization algorithms based on centroid in wireless sensor networks. In *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, pages 3216 –3219, april 2012.

[42] Zhonghua Zhao, Wei Huangfu, and Linmin Sun. Nssn: A network monitoring and packet sniffing tool for wireless sensor networks. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*, pages 537 –542, aug. 2012.

Appendix A

Injector rate [s]	Packets lost	No. packets	Loss %
2	5	1495	0.33%
1.5	0	1598	0.00%
1	1	1520	0.07%
0.5	5	1457	0.34%
0.25	24	1603	1.50%
0.2	15	1450	1.03%
0.1	27	1529	1.77%
0.05	63	1569	4.02%
0.025	134	2031	6.60%

Table 8.1: Evaluation of the sniffer capacity with injection with traffic of 50 packets/s

Injector rate [s]	Packets lost	No. packets	Loss %
2	2	1879	0.11%
1.5	9	2633	0.34%
1	1	1951	0.05%
0.5	9	2120	0.42%
0.25	15	1990	0.75%
0.2	21	2606	0.81%
0.1	48	2051	2.34%
0.05	48	2076	2.31%
0.025	109	1708	6.38%

Table 8.2: Evaluation of the sniffer capacity with injection with traffic of 68 packets/s

Injector rate [s]	Packets lost	No packets	Loss %
2	0	3072	0.00%
1.5	1	2608	0.04%
1	4	3590	0.11%
0.5	n/a	n/a	n/a
0.25	n/a	n/a	n/a
0.2	n/a	n/a	n/a
0.1	n/a	n/a	n/a
0.05	n/a	n/a	n/a
0.025	n/a	n/a	n/a

Table 8.3: Evaluation of the sniffer capacity with injection with traffic of 100 packets/s