ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing Group

Master Thesis

Wireless Multihop Communications for First Responder Connectivity

Johannes Geissbühler
johannes.geissbuehler@gmail.com

Prof. Dr. Roger Wattenhofer
Distributed Computing Group

Advisor: Dr. Michael Souryal

Dept. of Computer Science
Swiss Federal Institute of Technology (ETH) Zürich
Summer 2006

Abstract

First responders, using current radio communication technologies, entering a large building can lose radio communications with their incident command due to propagation loss of the radio signal with distance and obstruction. Multihop wireless networks, using intermediate nodes to relay the signal over multiple, shorter hops from source to the destination, have generally been proposed as a means of extending radio coverage.

In this thesis we demonstrate real-time deployment of a multihop network to maintain communications between incident command and a first responder entering a building. The system continuously assesses the channel and determines when a new relay needs to be deployed. As the first responder moves deeper into a building, a communications link is automatically configured over which data traffic is transmitted end-to-end.

In addition to the deployment algorithm we present how power control of the relay nodes can be used to maintain good radio link quality. To access the network we present a routing protocol, which is particularly adapted for this type of fast changing ad-hoc network.

The prototype system of this thesis is implemented on 900 MHz Crossbow mica2 motes supporting periodic transmission of sensor values (e.g., vital signs of the first responder) to the incident command as well as two-way text messaging and Radio Frequency Identification (RFID) information captured by the first responder. We evaluate the prototype system inside a common American five floor office building. The achieved results show that the proposed system is well suited for first responder connectivity.

Zusammenfassung

Einsatzkräfte, die mit Funkgeräten der aktuellen Technologie ausgerüstet sind, riskieren beim Betreten von grossen Gebäuden das verlieren der Funkverbindung zur Einsatzzentrale. Die Verbindung kann infolge der Abschwächung des Funksignals durch Distanz und Hindernisse durchtrennt werden. Als allgemeine Lösung um das Funksignal über grössere Distanzen zu propagieren, wird heutzutage der Ansatz von Multihop Wireless Netzwerken vorgeschlagen. Multihop Wireless Netzwerke benutzen zwischen Ausgangspunkt und Ziel liegende Relay-Knoten, die das Signal über mehrere kürzere Netzwerk-Hops zum Ziel weiterleiten.

In dieser Arbeit zeigen wir einen Ansatz auf, welcher es erlaubt in Echtzeit ein multihop Netzwerk aufzubauen. Dieses Netwerk ermöglicht die Kommunikation zwischen den Einsatzkräften, die ein Gebäude betreten, und der Einsatzzentrale in zuverlässiger Weise. Das System beurteilt laufend die Qualität des Funkkanals und entscheidet zu welchem Zeitpunkt ein weiterer Relay-Knoten platziert werden muss. Während die Einsatzkraft sich immer weiter in das Gebäude hineinbewegt, wird die Verbindung, welche zweiweg Datenkommunikation mit der Einsatzzentrale ermöglicht, laufend angepasst und aufrechterhalten.

Neben dem Algorithmus zum platzieren der Relay-Knoten zeigen wir auch wie das Regulieren der Sendeleistung dieser Knoten benutzt werden kann um gute Linkqualität beizubehalten. Um das erstellte Netzwerk schlussendlich auch benutzen zu können, präsentieren wir einen speziell auf diesen Typ von ad-hoc Netzwerken zugeschnittenen Routing-Algorithmus.

Der Prototyp dieser Arbeit ist mit 900 MHz mica2 Crossbow motes implementiert und erlaubt periodisches senden von Sensorwerten (z.B. Hertzschlagrate von der Einsatzkraft) zu der Einsatzzentrale. Des weitern werden zweiweg Textnachrichten und Radio Frequency Identification (RFID) Datentransfer unterstützt. Das Prototyp System wird in einem repräsentativen fünf stockigen amerikanischen Bürogebäude getestet. Die erlangten Resultate zeigen, dass das in dieser Arbeit aufgezeigte System geeignet ist um die Funkverbindung von Einsatzkräften auch in grossen Gebäuden aufrecht zu erhalten.

Acknowledgements

This thesis is the result of a stage at the National Institute of Standards and Technology NIST in Gaithersburg, MD, USA. I'd like to cordially thank to all people who made this great experience possible.

First to **Prof. Dr. Roger Wattenhofer** at ETH for his flexibility. Without his help, this internship and thesis would not have been possible.

My host at NIST **Dr. Nader Moayeri** for giving me the opportunity of writing my master thesis at the National institute of Standards and Technology.

Many thanks go to **Dr. Michael Souryal** for his continuing support, suggestions and interesting conversations.

I also want to commemorate my supervisor **Dr. Leonard Miller** who tragically passed away on June 7, 2006. He was a great person who helped my in many ways during my stay in the United States.

Last but not least I want to thank all other persons who supported my in any direction in writing this thesis. There were so many it's impossible to write down all their names. I really appreciated the help of everyone.

Contents

Chapter 1

Introduction

1.1 Motivation

Reliable communication is a critical resource for public safety operations including emergency response. A common scenario for responding to indoor emergencies, such as a building fire, is for an incident command center to be established outside of the building and for first responders to enter the building to conduct search and rescue. When the building is large, incident command can lose radio communications with first responders due to severe attenuation of the signal, thereby losing the ability to track the status of personnel and to coordinate units. One approach to alleviating the risk of communication loss (or outage) is to use multihop wireless communications to extend coverage. In the event that the signal strength on a single hop is too weak to sustain a desired quality of service, relays deployed between the end points can serve to extend the range of communication by repeating transmissions in a coordinated manner.

Much work has been done previously in the areas of medium access and routing for ad-hoc networks and multihop communications. The key new challenge in the context of emergency response is the task of real-time network deployment - that is, determining where relays need to be deployed, and doing so in real time with a level of automation that minimally impacts the search and rescue tasks of the first responder. Furthermore, the deployment function must operate concurrently with and not interfere with the MAC and routing protocols that facilitate reliable delivery of application traffic.

1.2 Assignment

The task of this thesis is to develop a proof of concept prototype of a first responder communication application which is based on a wireless multihop network. The scenario consist of an incident command located outside a building and a first responder moving into this building. In addition to a reliable two way communication channel the incident command wants to get vital signs and location information from the first responder. To build this prototype we have to find a suitable hardware platform and investigate the specific radio propagation attributes together with other hardware related constraints. Given this hardware platform we have to develop algorithms to perform real-time relay-node place-

ment and reliable end to end routing. We also want to investigate the possibility to use power control on the transmitter to obtain high quality radio links. Based on this wireless multihop communication layer we have to implement our application which uses this network and allows ongoing communication between the incident command and the first responder.

As we want to have a running prototype, testing plays an important rule. Given that our focus lies on dynamic indoors scenarios it's almost impossible to simulate such environments in a realistic way, therefore we have to test the prototype in different real world situations to find suitable parameters for the algorithms and to verify the functionality of the concepts. An important point to verify is in any situation the reliability of the prototype as we are dealing with vitally important systems for first responders.

1.3 Related work

Much scientific work has been done in the area of ad-hoc networks especially multihop ad-hoc networks. There exist thousands of papers addressing a huge variety of themes in wireless ad-hoc networks - the most important topic is most likely routing. This work has largely been done in a very theoretic way whereas we want to focus on work which has been used on real hardware. Though there exist only very few applications and products which are used in the everyday life using wireless multihop ad-hoc networking. The 802.11 ad-hoc mode is not really reliable, does not scale very well and never had its big breakthrough. Besides 802.11 cards there are a few military products available, for example SpearNet Team Radio [1], which use multihop protocols to communicate.

The big upcoming field is sensor networking and mesh networking in general. Concerning sensor networks, a lot of work in multihop protocols in all variety has been published already. A good overview over the field of sensor networks is given in [2]. Other examples are [4], which proposes incorporating link quality metrics into the routing protocol for sensor networks and [3], which compares and analyses different routing protocols. Also in the field of radio signal propagation and radio signal strength distribution has been done a lot of work. The work in this area is mostly used for localization purpose. A paper dealing with signal strength and antenna patters is [6], the localization subject can be found in [5]. Also the idea of using power control has already been discussed. Power control used for localization is discussed in [7] and to save power in [8].

But what makes all work very different from our project is: they are all based on sensors which are placed at a fix position before using the network. That means the dynamic real time deployment aspect is totally missing in traditional sensor network applications. There exist indeed modern Mesh networking application, mostly based on the 802.15.4 ZigBee standard, supporting dynamic handover between different Backbone nodes, but there again the backbone nodes are not ad-hoc anymore and are also installed in a preset way before using them.

There are also some approaches which try to simulate ad-hoc networks, instead of testing them in real test beds. The main difficulty in simulating an indoor RF channel is the strong dependence of the received signal on the layout of the building (i.e. multipath channel). In particular, all walls, windows, and other objects that affect the propagation of RF waves will directly impact the signal strength and the directions from which RF signals are received. There

exists raytracing tools, for example WISE [9], which try to predict the received signal in an indoor RF channel. Because these tools are not that easy to use and still relay on too much parameter, most experiments are still done with large sensor test beds. But again most of the big sensor networks test beds are deployed in nature - that means they focus outdoor radio signal strength propagation, often even in line of sight situations and are fix installed.

To summarize this short overview we can say that a lot of theoretical work in the area of ad-hoc wireless multihop communications already exist. Considering the different challenges of our planned prototype all isolated, there are miscellaneous suggestions how to solve them separately but not how to bring them to work in combination.

Different situation with our main task: the real-time deployment of relays has hardly any been discussed. Especially a working prototype, which uses this real time deployed ad-hoc network, has not been implemented so far. This is exactly what we want to do in this thesis.

1.4 Overview

The rest of this document is structured as follows: In chapter 2 we present our selection of the hardware we are using to implement our prototype. Together with the hardware we also expose our software choices. In chapter 3 we present the results, we obtained while exploring the hardware properties and constraints which concern our project. In chapter 4 we use the results from the previous chapter and present, based on them, our dropping algorithm. The implementation details are explained in chapter 5 whereas the results are evaluated in chapter 6. In chapter 7 we conclude this thesis by summarizing the derived outcomes. Also an outlook of future work and unsolved problems is given there. In the Appendix the most important Interfaces and Message types are printed out in detail.

Chapter 2

Hardware and Software Decisions

The base scenario for our Wireless Multihop Communications for First Responder Connectivity prototype consists of following parts: An incident command node, which is normally located outside a building and a first responder node, which is carried by the first responder into and about the building. The first responder node is equipped with a sensor board to periodically send vital signs to the incident command. Additional to the sensor board the first responder node also includes a RFID reader to send location information to the incident command. The location details are provided on RFID tags installed in the building itself. Besides the sensor values and the location information a way to communicate between the incident command and the first responder has to be provided.

In this section we present the hardware and software decisions we had to take in order to achieve our goal. Especially the hardware decisions have a huge influence on the ongoing project and the final prototype. There are many choices in the hardware and software section which have not been taken at the begin of the project but rather evolved during the project. Nevertheless we present all our decisions here as they ended up being in the final prototype.

2.1 Hardware

In the hardware section we want to present our assortment of hardware to implement our prototype. Because the selection of the hardware narrows the room for options very hard and also is hard to reverse during the project it was quite hard to find the perfect choice. In most cases also the perfect, full adjusted hardware is not available. Therefore the choice was often influenced by cost, availability and last but not least the publicity of some manufacturer and parts.

2.1.1 Platform

The first and probably most drastic decision was to choose the hardware platform of the radio itself. The hardware nodes must be capable to send in ad-hoc

mode and to act as a relay. Furthermore it's important that we can program the behavior of the node itself in a convenient way. Also we want to be able to make changes even down in the MAC layer of the nodes.

Already the wireless ad-hoc mode criterion narrows the available hardware products very hard. Basically there are three different categories available: 802.11 wireless cards, sensor nodes and mesh network components. The 802.11 cards in the ad-hoc mode provide almost no flexibility to make changes in the MAC layer. Also the 802.11 ad-hoc mode still seems not to be beyond doubt of reliability and scalability. Using standard wireless cards, every node gets quite expensive because you need a laptop or PDA together with every card. So standard wireless cards were pretty soon out of discussion.

There are more and more vendors who provide out of the box Mesh network nodes like [10], mostly based on the 802.15.4 ZigBee protocol. The big disadvantage of this Mesh nodes are, that they often only provide a serial data interface and no possibility to program the behavior of the node itself. Thus also these out of the box network nodes products are not a valid option for us.

This leads to the only remaining category: sensor nodes. Sensor nodes are a strong growing domain particularly because the hardware is getting cheaper and smaller very rapidly and in the same time is becoming more powerful. Although sensor nodes are mainly designed for stationary, energy efficient and low duty work and not for real-time computing, they provide more or less everything we need. They are programmable, even the MAC Layer, provide ad-hoc wireless radio and sensor boards, are widely used for research projects, are small and not too expensive and can be battery-powered, which means they are mobile. We decided to use sensor motes developed by Crossbow Technology Inc. [11]. There the decision fell to the 900 MHz mica2 motes and not the 2.4 GHz micaz ZigBee motes because the 900 MHz Frequency band provides better radio propagation inside buildings.

The mica2 motes feature the following basic data:

Processor:	8 bit ATMega128L processor running at 7.37 MHz
Memory:	128 kB program memory, 4 kB SRAM, 512 kB EEPROM
Radio Interface:	ChipCon CC1000 [12] chip sending at a frequency of 433 or 900 MHz respectively, allowing data rates of up to 38.4 kbps
Power Source:	$2 * AA$ (1.2 V)
Size:	58 x 32 x 7 mm (without batteries)
Weight:	200 g (with batteries)
Misc.:	3 LEDs (green, yellow, and red), 51-pin expansion connector allowing to connect external peripherals like sensor boards

More details are provided in the Crossbow Datasheet [13]. A picture of a mica2 mote can be seen in figure 2.1

2.1.2 Display and Interface

On both end nodes, the incident command and the first responder node, we need to have the possibility to display information and to provide a way to enter messages. On the incident command side we use any standard Laptop,

Figure 2.1: Mica2 sensor mote without antenna.

which provides an USB interface over which we connect the incident command mica2 through the MIB520 [14] USB Gateway with the laptop.

On the first responder side it gets a bit trickier. Because the first responder needs to be very agile, a smaller display and input device is needed. Therefore we use a standard IPAQ as the display and input device for the first responder. Since the IPAQ is not able to provide the MIB520 USB Gateway with power, we need a MIB510 [15] Serial Gateway, which connects the first responder mica2 mote over the RS-232 serial interface with the IPAQ. The MIB510 gateway is able to take its power from the battery pack of the connected mote whereas the MIB520 gateway has to be powered over the USB cable. The MIB520 and MIB510 gateways can be seen in figure 2.2.

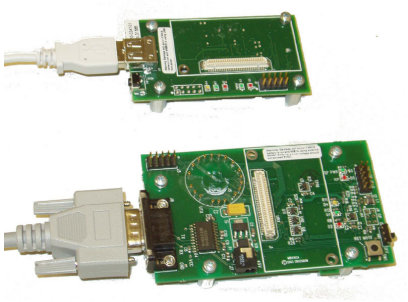

Figure 2.2: MIB520 USB (above) and MIB510 serial interface Gateway (below).

2.1.3 RFID

To provide the incident command with accurate location information we equip the first responder with a RFID reader. With this RFID reader the first responder then reads out RFID tags, which are installed in the building and contain actual location information. We made first tests with a SkyeTek [16] SkyeModule M8 [17] reader and ISO-18000 tags. We decided to do our first test with this SkyeModule M8 because it allows with its strong power and the big antenna, what results in sending power of up to 27 dBm, to read the tags even in a range of up to three meters. A picture of the M8 reader and an ISO-18000 tag can be seen in figure 2.3.

The SkyeModule is connected to the first responder IPAQ over the RS-232 interface. To provide wireless power, we attached the reader to a standard 9 volt DVD battery. The frequency of the reader can be set between 860 and 960 MHz

therefore there is enough margin in the frequency band to avoid interferences with the mica2 mote whose transmission frequency can be set between 903 and 927 MHz.

First tests with the SkyeModule M8 reader were quite successful, although the theoretical range of three meters was in practice at most two meters. We also realized that the ISO-18000 tags are much better read than the also supported EPC tags. With the M8 reader it was possible to pass the RFID tags attached to the wall in a normal way while the reader captured the location information stored on the tags.

As soon as we also wanted to send the location information to the incident command over the mica2 motes, we realized that the radio transmission of the mica2 mote was highly disturbed. We tried many things to avoid these interferences: Setting the reader and the mote at frequencies with the highest gap between them, switching on and off the frequency hopping of the reader and also tried different RFID tag types (they are read out in a different manner). Even analyzing the signals on the spectrum analyzer showed, that normally the two signals should not interfere with each other. The mica2 transmission signal as well as the reader signal were very sharp and did not interfere on the spectrum analyzer. Finally analyzing the whole spectrum from 0 Hz to 1 GHz showed the cause: The RFID reader's printed circuit board produces a very strong noise in the very low frequency area which seems to interfere with the mica2 mote board and disables somehow the mica2 mote from sending out messages.

As a next step we tried to avoid all the hassle we had with the SkyeModule M8 reader and tested a new ACG Dual ISO CF Card Reader Module [18]. A picture of this reader can be seen in figure 2.3. This RFID reader fits over a CF to PC card adapter in the PC Card slot of the IPAQ. Over this connection the power is directly taken from the IPAQ which results in a much smaller and lighter equipment for the first responder. This reader uses, different from the M8 reader, 13.56 MHz as the transmit frequency which is far away from the 900 MHz of the mica2 mote. The ACG reader supports only ISO1443A MIFARE tags.

The negative aspect of this reader is that due to the low power of the reader you need to go as close as three centimeters into the range of the tag to read it. That means you almost have to touch the RFID tag with the antenna in order to read it. Taking into account that we produce only a prototype, we decided that this is suitable for our purpose. Tests with this reader together with the mica2 mote were successful and showed no interference.

2.1.4 Antennas

The Crossbow mica2 900 MHz motes are by default delivered with a standard one-quarter wavelength monopole antenna 10 cm in length, which is connected to the printed circuit board of the mica2 mote over an MMCX connector. Therefore antennas normally are not an issue when dealing with Crossbow motes. But during some tests we realized some very strange behavior of the nodes. After very time consuming debugging we found out, that the motes are measuring very different Received Signal Strength Indication (RSSI) values and also the transmission power between the different motes varies widely. Results of this measurements can be seen in table2.1. To obtain the values from this table we put each mote in a distance of 2.5 meters to the same measurement station and

Figure 2.3: SkyeModule M8 reader with battery pack (left), and ACG Dual ISO CF Card reader (right) with the corresponding RFID labels.

Mote Nr	RSSI at gateway 1	RSSI at gateway 2
2	-95.9	-77.3
3	-81.4	-76.9
4	-51.0	-50.6
5	-50.8	-50.5
7	-51.0	-51.0
8	-52.4	-50.9
9	-50.4	-50.5
10	-51.2	-50.7
11	-51.3	-52.2

Table 2.1: RSSI values with default Crossbow monopole antenna obtained on symmetric link test.

sent 1000 packets in each direction measuring the RSSI on the other side. That the measurements are independent from the voltage of the battery we used on both sides a MIB510 USB gateway, which powers the mote with stable 3.3 volts.

After more intensive tests we found out that the MMCX connector is the weak point. It seems that the MMCX connector provides a very lose connection between the antenna and the board itself, which can result in high loss of sending and receiving power. Another negative effect of this MMCX connector is that the antenna often falls from its upright position, which also affect the radio signal strength as we will see in section 3.6.

Having realized that the MMCX connector is the weak point we searched for alternative antennas, which can be soldered directly onto the printed circuit board. We found two alternative antennas from Linxtechnologies [19]: A very small embedded antenna ANT-916-JJB [21] and a normal one-quarter wavelength monopole antenna ANT-916-PW [20]. The JJB antenna can very easily be soldered to the board and due to its very small dimension it's also very convenient in practical use. The PW antenna is much harder to solder, because we had to stripe off the coaxial cable and also solder the ground carefully to the board. In practical usage the PW antenna is also very inconvenient because it's not so easy to attach the antenna onto the board. We did the same test measurements as we did to find out the difference between the Crossbow default

Antenna type	Frequency	RSSI at gateway 1	RSSI at gateway 2
default monopole	916	-50.1	-49.9
default monopole	927	-49.4	-49.3
linx monopole	916	-49.6	-49.8
linx monopole	927	-49.3	-49.9
linx JJB	916	-52.5	-51.9
linx JJB	927	-53.4	-53.4

Table 2.2: RSSI values with different antennas obtained on symmetric link test.

antennas again, to compare the three different antenna models. In addition to the previous test we were also interested in how much influence the frequency has on the RSSI values. The Antennas are optimized for the frequency of 916 MHz thus we tried exactly 916 MHz and the furthest possible away from that, what is 927 MHz. As already the datasheet stated the small JJB antenna loses gain if the frequency diverges from the 916 MHz whereas the monopole antennas are less sensible. The RSSI results can be seen in table 2.2.

Considering the small power loss of the embedded antenna compared to the default antenna and the gain of the certitude that we don't experience weak connections any more and that the antenna is always in the right position, we decided to changeover our mica2 motes to the embedded JJB antenna. Only as the incident command antenna we are using the big ANT-916-PW, because it can easy be fixed onto the MIB510 USB gateway, which we are using at the incident command. How the JJB and the PW antenna look mounted on the motes can be seen in figure 2.5.

Figure 2.4: Standard Crossbow monopole antenna (left), Linxtechnologies PW monopole antenna (center), and embedded Linxtechnologies JJB antenna (right).

2.2 Software

In the software section we want to point out which software in what configuration is used to make this prototype work as a single unit.

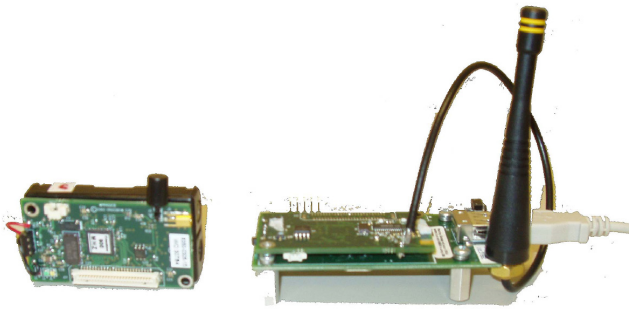

Figure 2.5: Embedded JJB antenna soldered on mote(left), and PW monopole antenna used at the incident command(right).

2.2.1 Platform

Choosing mica2 motes as the hardware platform was also choosing TinyOS [22] as the operating system because TinyOS is the usual operating system running on the nodes. At the moment there exists TinyOS 1.x and 2.0 beta versions. Because the 2.0 version is still in the beta phase and the 1.x version is still the common version used in the community we decided to use TinyOS 1.15. TinyOS was originally developed at the University of Berkeley. It is an event based operating system designed for embedded networked sensors. Since it is released under an open-source license, it is available for free and it was ported to other embedded platforms. The TinyOS software package comes with the most important library modules (network stack, hardware drivers, ...) and the tools which allow the mote to communicate with a Java application running on a computer. Due to the very limited resources available on most embedded sensor devices, TinyOS 1.15 has some heavy constraints:

- There is no concurrency in TinyOS, i.e. only one task (this is how pseudo threads are called in TinyOS) can be running at the same time. This task can not be interrupted by another task. The only exception of this rule are hardware interrupts which may interrupt the current task at any time [1]. Nevertheless, it is possible to handle multiple activities in parallel: The current task can create new tasks which are placed into the TinyOS task queue. This queue is processed in FIFO [2] order when the current task finishes. There is no possibility to give different task different priorities. This means that sometimes it makes the timing very hard because you have no guaranty how long it will take till an important, time critical task is processed. Switching to TinyOS version 2.0 would give the programmer much greater control over the task queue. The advantage of the restriction to only one concurrent task is that the compiler can detect data races [3] at compile time. The disadvantage is that the program structure becomes more complicated because bigger jobs need to be split up in multiple small

[1] Except when using the *atomic* keyword, see [24].
[2] First in, first out
[3] Data races may occur when a variable is accessed by a hardware interrupt handler and by a task of the module simultaneously.

tasks, disguising their correlation. This makes TinyOS applications harder to write and difficult to understand.

- TinyOS does not support dynamic memory allocation. Everything is allocated on the stack; there is no heap in TinyOS. This restriction greatly influences the design of TinyOS applications: Data structures must be initialized to their maximum size at compile time and therefore probably reserve too much of the limited memory of small sensor nodes. However, due to this restriction, a TinyOS application may hardly contain memory leaks. What makes this constraint especially painful is the very small available stack of only 4 kB SRAM. A dynamic memory module, recently developed by the DCG [23] group, can bypass this constraint in the future.

- The size of radio messages sent using the TinyOS library is by default limited to 29 bytes of payload. This implies that bigger data packets must be split into smaller parts. Splitting, transmitting (including possible retransmission of lost or corrupted packages) and recombining has to be done manually, there is no library offering this service.

The programming language of TinyOS is a descendant of C called nesC [24] . It uses the C syntax and adds some new constructs. The most important concept of nesC is the separation of construction (the implementation) and composition (the wiring): A nesC application consists of at least one component (also called module) and one configuration file. The modules contain the actual implementation while the configuration specifies how the modules are wired together. Communication between modules is done using interfaces. These interfaces are - opposed to interfaces in popular languages like Java - bidirectional. For example, the interface of a timer may offer commands to start and stop the timer, but it forces the user of the interface to supply an implementation for an event executed when the timer fires.

Good documentation to start with TinyOS can be found in following tutorials and papers: [25] [26] [27] [2]. If any unsolvable problem occurs it's always a clever decision to search through the TinyOS mailing list archive if someone else already experienced the same problem.

The most convenient way to write TinyOS code and also to install the whole programming environment is to use the TinyOS Eclipse plugin [28] also developed by the DCG group of ETH Zürich.

2.2.2 Display and Interface

To interact with the user on both sides, the incident command and the first responder side, we decided to use Java as the programming language. On the first responder side the hard part was to find a Java Virtual Machine, which runs on the IPAQ. With Mysaifu JVM [29] we found a Java Virtual Machine which runs on Windows Mobile 2003 for Pocket PC(Pocket PC 2003) operating system. Mysaifu is free software under the GPLv2 license and easy to install on the IPAQ. The bridge between the serial port of the MIB510 and MIB520 (software emulates a com port on the USB interface) builds the Serialforwarder program which comes already with the TinyOS distribution. Important is that the Serialforwarder application has to be adapted that it uses acknowledges on this link, otherwise packet loss on this link is possible. On the IPAQ the

Serialforwarder has also slightly to be adapted, because the Mysaifu JVM does not follow the JAVA rules for the COM-port descriptors. For many experiments we decided to use a MySQL database to store the results for easy further use.

2.2.3 RFID

Fortunately both RFID readers do not need any special drivers or special software. They both can be addressed over their RS-232 serial interface which is not too complicated with Java. But because the RFID read events can occur asynchronously, the serial interface has to be controlled by a thread. The hardest part is to find the correct commands to operate the reader - the documentation is mostly very poor.

2.2.4 Overview

In figure 2.6 all hardware and software parts are showed how they interact with each other.

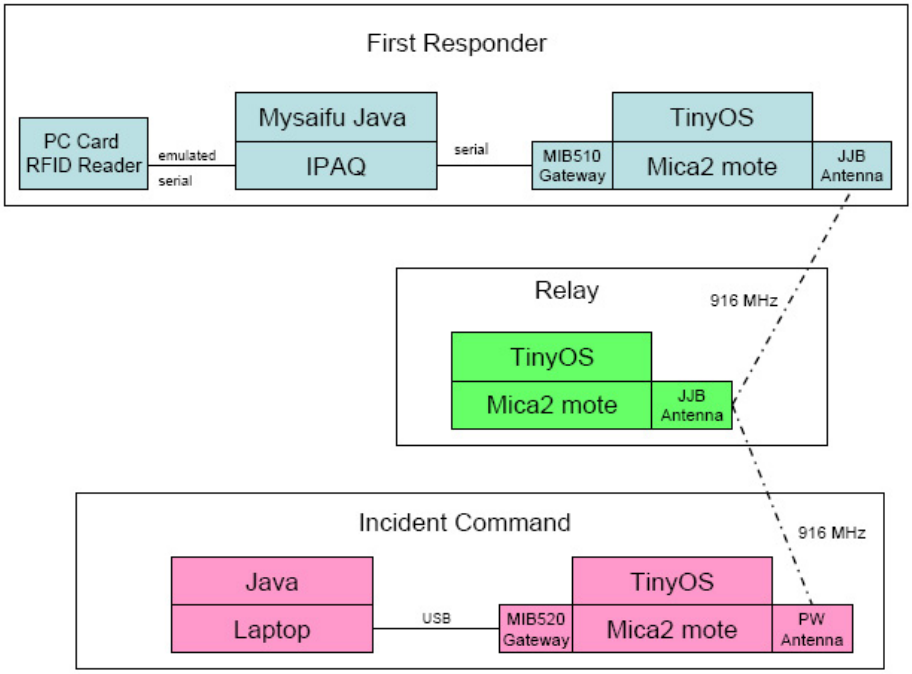

Figure 2.6: Hardware and Software components - how they interact with each other.

Chapter 3

Exploring the Hardware Constraints

In this chapter we present the result we obtained while exploring the mica2 hardware. Our focus was laid to find out if it's possible to do real-time radio channel quality assessment.

3.1 Packet Drop Threshold

First thing we wanted to find out was, if we can use the measured signal strength at receiver as a useful predictor of the channel quality and therefore also the link reliability. On the mica2 motes it is relatively easy to obtain the RSSI [1], therefore link assessment based on RSSI values would be an easy way to go.

To find out if there exists a kind of threshold we placed motes at different distances from a base station and let them send a large amount of packets. We also varied the TX power to obtain the whole range of possible RSSI values. The received packets at the base station we stored in a database and plotted the packet reception rate against the average received signal strength in dBm.

Analyzing the corresponding graph in figure 3.1 we were quite happy. The graph shows quite clearly that there exists a sharp threshold at -95 dBm. The sharp threshold allows us to set a clear criteria of how long the channel is still in good quality and when it isn't anymore. On the other side the very steep drop of the packet success rate around -95 dBm can also be dangerous because there is almost no buffer zone and the radio channel can become very quickly useless.

3.2 Effect of Mobility

Because first responders normally move around in the building more or less fast, we wanted to find out the effect of mobility onto the RSSI measurement. It's quite hard to separate the influence of changing the location (multipath fading)

[1] Received Signal Strength Indication. RSSI is a measurement of the received radio signal strength (energy integral, not the quality) value of a received packet.

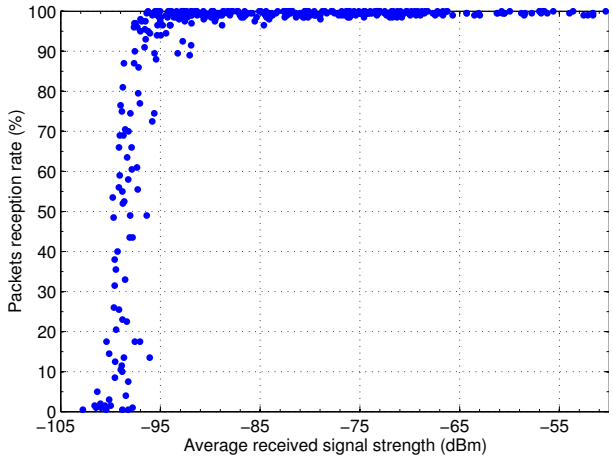

Figure 3.1: Average RSSI against packet reception. This measurement was obtained on 100 different links - on each link with 6 different power levels. The RSSI average is based on 200 packets.

from the influence of having the sender in motion itself. To minimize the influence of changing the location but still having movement, we mounted a mote onto an old turntable and made measurements with different speeds. Unfortunately already the different positions on the turntable without any movement result in a variation up to 15 dBm as you can see in figure 3.2. But comparing the RSSI measurements in the same adjustment with different speeds of the turntable shows that the motion itself makes no difference. The results of this measurement can be seen in figure 3.3.

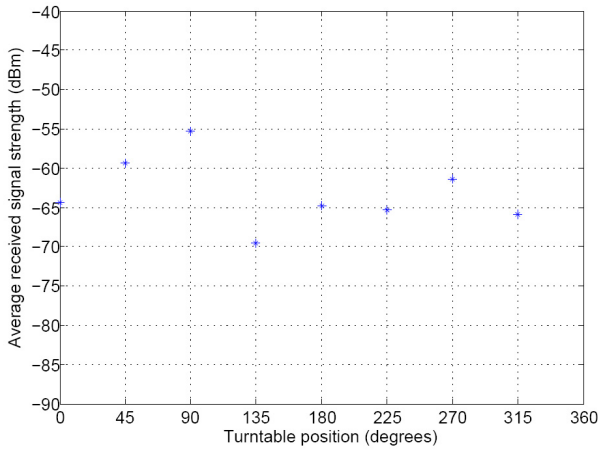

Figure 3.2: RSSI measurements on different positions on the turntable without turning.

As a second test we wanted to see the evolution of the RSSI when a first

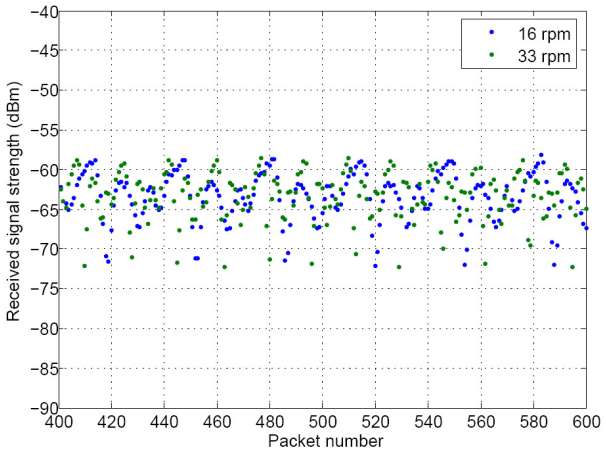

Figure 3.3: RSSI values on the turntable with the turntable turning on two different speeds.

responder walks along an aisle. For this test we mounted the mote onto a little LEGO car and attached a chord to this car, which is constantly pulled along the hallway by the turntable. The alignment with the LEGO car and the turntable is showed in figure 3.4. With this alignment we realized a constant movement of about 0.3 meters per second. The mote then sent packets to the inert base station every 20 milliseconds while moving along the hallway.

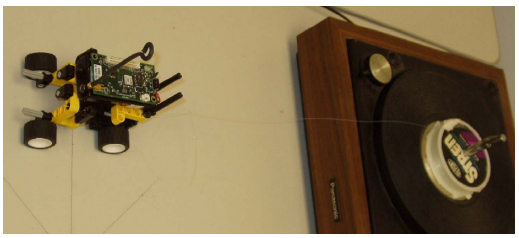

Figure 3.4: Mica2 mote on LEGO car which is pulled by a turntable.

The graph we produced from the caught packets is showed in figure 3.5 and could be stolen from a teaching book. The graph shows very well the influence of multipath fading on the RSSI values. Analyzing the data we can see that the signal strength varies up to about 20 dB over distances on the order of 10 cm ($\lambda/4$). This high variability on very small distances we have to take into account while designing our dropping algorithm.

3.3 Link Symmetry

Till now we did the RSSI measurements only in one direction of the radio link. But we wanted to now, if we can conclude only from one way RSSI measurement to symmetric link quality. To find out the coherence between the RSSI values on

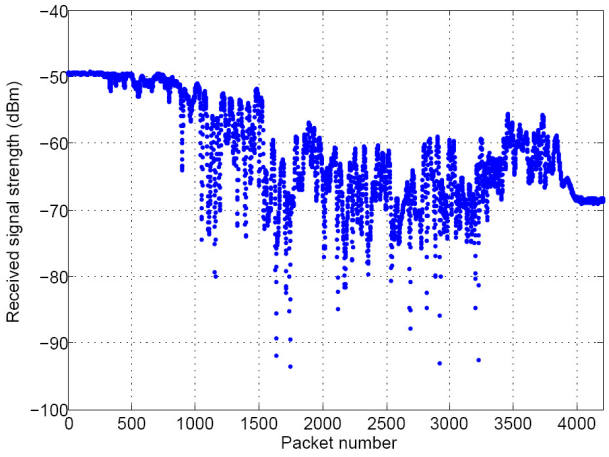

Figure 3.5: RSSI values measured on continuous movement with 0.3 m/s along a hallway. The sending frequency was one packet every 20 ms.

both sides we did following two experiments: First we used the CC1000RadioAck MAC layer which acknowledges every packet with a short bit sequence. This small bit sequence is long enough to gather also a RSSI value on the sending mote. Therefore we sent 1000 packets from one mote to the other and saved the RSSI values on both sides and lather on compared the two gathered RSSI values for each packet.

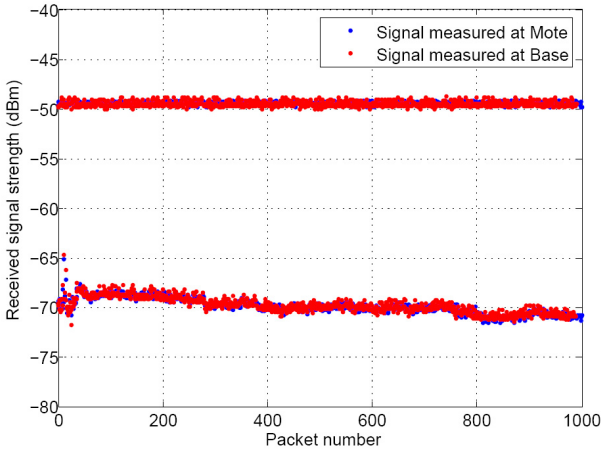

Figure 3.6: RSSI measured on both sides, Base and Mote, to verify link symmetry. The figure shows two different measurements.

As a second experiment we sent 1000 packets in both directions without the acknowledge MAC layer and also compared the RSSI values of this measurement. Looking at the result we can conclude: RSSI measurement in both directions of a link were found to be within a very small margin of one an-

other. In absence of locally varying interference, the results imply symmetric link quality.

3.4 Power

As the RSSI measurements values seem to become very important for our dropping algorithm we wanted to deeper investigate the correlation between power and measured RSSI value on the receiver. Power is interesting in two ways. First we wanted to find out if the power of the batteries, which energize the motes, has any influence on the signal strength itself. Second the ChipCon CC1000 chip [12], which is the RF chip on the mica2 motes allows to set the RF power from -20 dBm up to 5 dBm. We also wanted to test the influence of the different transmit power settings on the measured RSSI value.

The datasheet of the mica2 motes says that the motes work properly between 2.7 and 3.7 VDC. To test the difference voltages we took three motes and a base station. One mote and the base station were external powered which results in a 3.3 V power source. For the two other motes we used batteries. One pair of battery was quite new which results in 2.9 V and the other pair was quite old with results in 2.4 V what is below the 2.7 V which are indicated in the manual. The three motes then sent packets at different power levels to the base station which measured the RSSI value. The measurements showed that the different voltages had absolutely no influence on the measured RSSI values as well on the sending mote as on the receiving mote. Even with the 2.4 V, which is below the guidelines of Crossbow, the measured RSSI values where still the same. This result is very enjoyable. It will make no difference if the motes have totally new batteries or older ones, or are even energized by a external power supply.

For the second test we wanted to go through all the 256 power levels of the CC1000 chip (the strength can be set by a value between 0 and 255). For this measurement we again had a base station and a sending mote which sent packets on all 256 possible power levels. The result can be seen in figure 3.7.

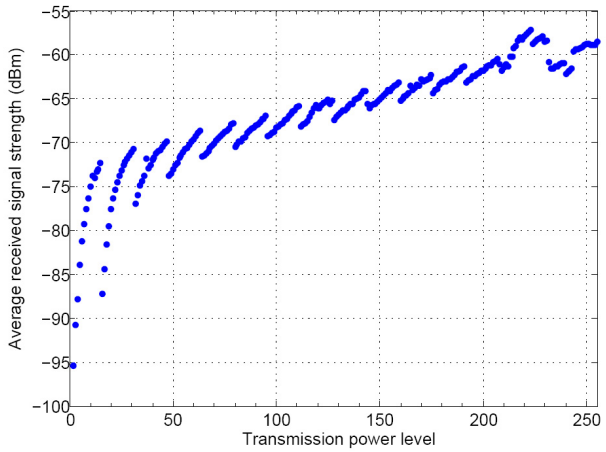

Figure 3.7: RSSI values in correlation of the transmitting power settings of the CC1000 chip.

What attracts attention is, that there is no monotonic increase of signal strength. So if one wants to adapt sending power in fine granularity one has to take care to pick the right values. Also the change in dBm in the first 10 power levels is much higher than later on. Also on the higher end of the power settings the result looks a bit strange, why is not really clear. But having the possibility to change the power of the radio signal in a range of almost 35 dBm gives us a good instrument to use power adaptation for link topology control.

3.5 Effect of Receiver Height

Because in our dropping algorithm it will be most likely that you drop the relays to the floor but you measure the signal strength at belt height we also wanted to see if dropping the relay on the floor will affect the link quality. To find out this, we measured the RSSI values at different locations on three different heights: directly on the floor, on 38 cm above floor and 120 cm above the floor. The results can be seen in figure 3.8.

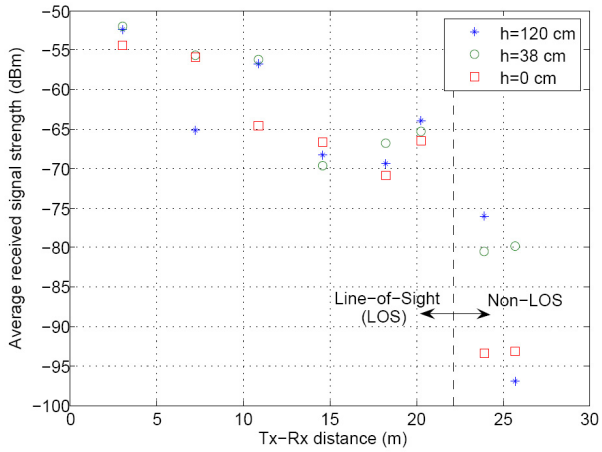

Figure 3.8: Difference in the RSSI value due to different height of the measuring mote.

Looking at the result we can see no persistent difference. The signal strength can change up to ± 10 dB from belt height to floor due to change in multipath profile. This means to us that we have build additional margin into the dropping threshold.

3.6 Antenna Position

We also wanted to test the influence of the position of the antenna. Because the connection of the default Crossbow one-quarter wavelength monopole (see also section 2.1.4 for more details on the antenna itself) is not very stable. It often occurred that the antenna did not rest in its upright positions. We turned the antenna in 45 degree intervals and measured the average RSSI value obtained on this antenna. The results can be seen in figure 3.9.

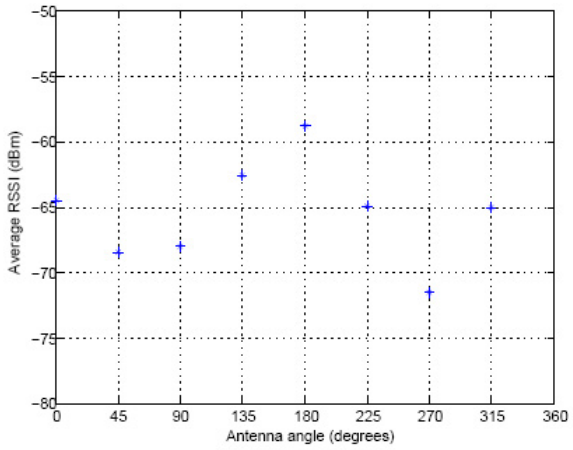

Figure 3.9: Difference in the RSSI value due to different position of the antenna - measurement done with the default Crossbow one-quarter wavelength monopole antenna connected over the MMCX connector.

As we can see the influence of the position of the antenna is up to 10 dBm. But we decided that for us this is not a big issue. In a final product you could always have more than one antenna and automatically use the one which gets the best signal quality (what is often done with wireless card antennas, which are integrated in laptops). For our later tests of the final prototype we tried to have the antenna always in the same position. Especially with the new embedded antenna (se again section 2.1.4 for details), which is directly soldered on the mote, the antenna stays always in the same position. With the embedded antenna it's more important that you place the relay motes in a way the antenna points against the ceiling an not in a horizontal way.

Chapter 4

Dropping Algorithm

Using the different results from chapter 3, we devised a parameterized dropping algorithm based on rapid probing by the first responder node. This Algorithm is explained in detail in this chapter.

4.1 Dropping Location

In this section we want to introduce the base algorithm, which decides where the first responder has to place a new relay node to keep up it's radio communication with the base station. First we explain the base algorithm, then we focus on MAC layer items and third we explain in detail the different parameters we have chosen for the algorithm.

4.1.1 General

The general deployment algorithm is quite straight forward and acts as follow: The first responder broadcasts every Δ seconds a channel probe. Any node in the network, who has itself connectivity to the base station, responds with a simple unicast acknowledge message to the first responder.

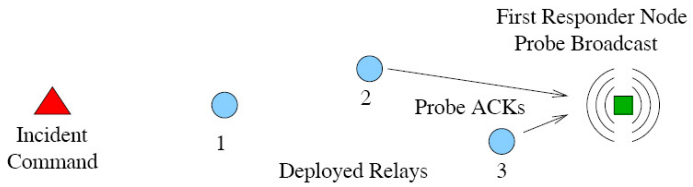

Figure 4.1: Broadcast channel probing performed by the First responder Node.

The first responder then measures the RSSI of each probe ACK it receives and keeps a store of the N most recent probe periods of each replying node. If an ACK message is missed from any replying node, a low RSSI value S_0 is placed instead in the store. A LED indication is given to the first responder to deploy a new relay when the average RSSI measurement over the last N probing

periods of all relays is below a threshold, S_{th}. In other words, letting $S_{i,j}$ be the RSSI measurement from the ith replying node during the jth probe period, deployment is triggered when

$$\max_i \left\{ \frac{1}{N} \sum_{j=1}^{N} S_{i,j} \right\} \leq S_{\text{th}}.$$

When a new relay comes into play, for the time till the data buffer is filled with actual measurements, single RSSI measurements are considered. If a relay does not reply during the last three seconds, the relay is considered as not existent any more and it's RSSI store is not considered anymore.

4.1.2 MAC

An important rule plays the MAC layer of the motes, because every node replies immediately to the broadcast probes of the first responder and we want to avoid collision of these packets as much as possible.

First some details about the normal TinyOS MAC Layer. We decided to use the standard, so called B-MAC, TinyOS MAC layer without acknowledgment, because they add to much overhead for our purpose. This B-MAC provides CSMA [1] with a random initial and congestion backoff window. As the channel frequency we have chosen 916 MHz because the antenna we have selected in section 2.1.4 brings best performance at this frequency. We also use Manchester encoding which results in a theoretical data rate of 19.2 Kbps. As mentioned above the biggest problem is collisions of the acknowledge packets if more then one relay is in range of the first responder and therefore more than one relay replies to the probing packets. We invested some time to analyze the behavior of the MAC layer and tried to optimize it. The default timing for a single packet transmission looks as figure 4.2 shows.

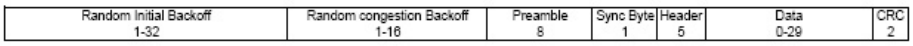

Figure 4.2: Default TinyOS B-MAC timing schema for sending one packet. Numbers are in bytes.

First we have 1 to 32 bytes random initial backoff before a first send attempt is started, then as long as the carrier is sensed busy a 1 to 16 bytes random congestion Backoff is inserted till the carrier is sensed free. Once the transmission starts an 8 byte preamble, a Sync byte, a 5 byte Header, 0 to 29 bytes of payload data and finally 2 CRC bytes are transmitted. With our data rate of 19.2 Kbps one byte corresponds to 416 us. That means standard initial backoff is between 416 us and 13.3 ms and standard congestion backoff is between 416 us and 6.7 ms.

[1] Carrier Sense Multiple Access (CSMA) is a probabilistic Media Access Control (MAC) protocol in which a node verifies the absence of other traffic before transmitting on a shared physical medium.

backoff	missed relay 1	missed relay 2	avg missed
32 / 16	44	37.7	40.8
64 / 32	27.0	37.7	22.8
128 / 64	24.3	19.7	22.0

Table 4.1: Comparing different backoff values - base station with 2 relays. Missed packets are out of 1000 received and returned packets.

backoff	missed relay 1	missed relay 2	missed relay 3	avg missed
32 / 16	88.7	90.3	67.7	82.2
64 / 32	39.7	39.0	29.3	36.0
128 / 64	29.7	41.7	20.7	30.7

Table 4.2: Comparing different backoff values - base station with 3 relays. Missed packets are out of 1000 received and returned packets.

For our dropping algorithm we decided to send channel probing packets about every 100 ms therefore we did some measurement with two and three relays in range and different initial and congestion backoff configurations. For this measurement we broadcasted 1000 packets in 100 ms intervals to two and three relays which responded immediately like in the real algorithm and then counted the missed packets at the base station.

As we can see from the result tables 4.1 and 4.2 the combination of 128 bytes (53 ms) for initial backoff and 64 bytes (27 ms) for congestion backoff window provide an adequate tradeoff between delay and throughput for a probe period of $\Delta = 100$ ms. These results were also verified by matlab simulations. Resulting for our prototype that the timing schema for one single packet transmission looks as figure 4.3 shows.

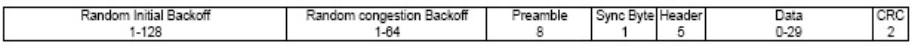

Figure 4.3: Our adapted TinyOS B-MAC timing schema for sending one packet. Numbers are in bytes.

Additionally to mention is, that we first tried to build our own custom random backoff on the application layer with an adapted backoff for every different node in addition to the standard backoff on the MAC layer. But the results were much worse than adapting directly the MAC backoff. This is probably because the timing on the upper layer is quite problematic because everything is handled in one big FIFO task queue and there the programmer has no influence on scheduling. So very small time intervals are probably not very accurate because a task can be delayed in the queue for some unpredictable time. That means programming with very short time intervals triggered by timers is probably not a good programming practice in TinyOS because the intervals won't be very accurate.

Δ	Probe period	100 ms
N	Averaging filter length	20
S_{th}	Threshold signal strength	-80 dBm
S_0	Default RSSI measurement value	-100 dBm

Table 4.3: Deployment Algorithm Parameters.

4.1.3 Parameters

Now as we discussed the base algorithm and the suitable MAC layer choosing the right parameters for the algorithm is very important. Table 4.3 lists the four parameters with the values we have to chosen for our prototype.

The probe period affects very directly the accuracy of the measurement of the channel. Theoretically we want to probe the channel as often as possible to have a very up to date measurement of the link. On the other side we don't want to probe the channel too often to leave some capacity of the channel for data transmission and also to avoid too many collisions from replying Ack packets. Another point is, that the motes do not have a very powerful processor, therefore too many probing packets also could fully load the CPU and the task queue what would result in an unpredictable workflow. After many different experiments and also taking into account our MAC layer we found a probing period of 100 ms reasonable.

The second parameter is the length of the averaging filter. The length of the averaging filter adheres very close with the probing period because the sum of the probing period and the averaging filter defines the time interval over which the channel probes contribute to the decision if the critical threshold is already reached. The tradeoff in choosing the filter length lies between having a very short and reactive filter and a longer more multipath fading tolerant filter, which brings the danger of being too slow and missing the right threshold point. Considering the results from section 3.2 we did several tests to find an ideal length of the filter which is not misled by local mulitpath gaps but also is reactive and conservative enough to find the place where to drop a relay.

The graph in figure 4.4 shows graphically some results we measured with a Δ of 100 ms and a threshold of -80 dBm. For this experiment we tried different filter lengths on different path and measured the RSSI after the algorithm indicated to drop the relay. The experiment showed that a filter length of 20 seems to be a good compromise. A length of five is too reactive what can be seen in the big variability; on the other side 40 is too slow which results in constantly too low RSSI values and therefore a radio link below our aimed quality.

The third parameter is the threshold itself. From the section 3.1 we know that the radio channel is turning bad at around -95 dBm. From section 3.2 and 3.5 we know that local multipath differences can be around 20 dBm and that due to the dropping of the relay we can also lose up to 10 dBm. In this section we have also encountered that our dropping algorithm has an accuracy of about 5 dBm. Although all this factors normally never add up we have to choose our threshold carefully and more on the saver side because reliability is still one of the most important attributes of the algorithm. Many tests showed that a threshold of -80 dBm is a good compromise between being on the safe

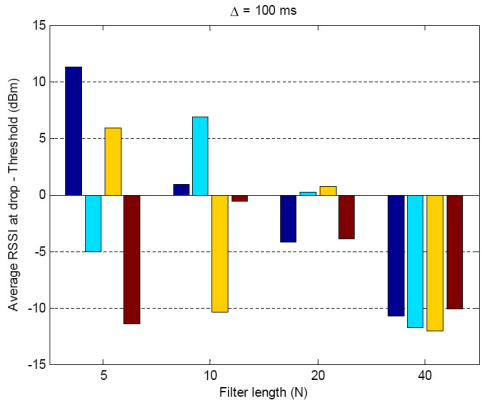

Figure 4.4: Divergence from the threshold with different filter length deployed with a treshold of - 80 dBm.

side and still having the relays not placed too close to each other.

The fourth and last parameter is the default RSSI measurement value of missed probe acknowledgments. Every time a unicast Ack packet from a relay is missed due to collision or because the signal was too weak this default value is inserted. The default RSSI value should be very low because a missed packet is a very bad sign and mostly because of a bad channel and not due to collision with other packets. So we decided to choose -100 as the default RSSI value which is below -95 dBm, the smallest value which is possible to measure as we learned in section 3.1.

4.1.4 Adaptive Probe Delay

In the section 4.1.3 we declared that the first responder sends a probing packet every Δ time, in our case every 100 ms. This makes complete sense when the fireman is moving around. But as soon as the fireman is not moving around anymore, it's a big overhead probing the channel every 100 ms. Therefore we decided to change the probing interval down to a Δ of every 500 ms when the first responder is not moving.

Our mica2 first responder mote is equipped with a sensor board which also delivers the vital sign measurement, the temperature and the luminosity. This sensor board also has a two dimensional accelerometer which can be used to detect the movement of the first responder. Because the accelerometer is quite sensitive and mainly is influenced by the alignment against the earth gravity field, it's almost impossible to find absolute values defining when the person is in motion and when not. The mechanism who detects if the first responder is in motion or not periodically measures the x and y axis acceleration and compares them to the previous value measured. If the difference is bigger than certain boundary value the person is considered moving otherwise stationary. Because again we want to be on the safe side and do not want to miss slow movement we are very conservative in stating that the first responder is not moving.

4.2 Local Placement

As we showed in section 3.2, signal strength can vary up to about 20 dB over distances on the order of 10 cm ($\lambda/4$) due to local multipath fading. To avoid placing the relay in a local deep fade, we added to the relay a visual placing help, which indicates over the three available LED's on the mica2 mote if the link to the next relay is smaller than a certain threshold. For our demonstrator we set the threshold to -90 dBm which is below the S_{th} threshold signal strength but over the -95 dBm which is the knockout value for the link. Normally the first responder does not need this indication but in bad luck, when the relay hits a deep fade, it's enough to push the relay only a few centimeters to the side.

The dropped relay displays its radio channel quality to the next relay as follow: Even after the new relay is dropped the first responder still sends the broadcast probing packets which are received by the new dropped relay but also still by the prior relay node. This prior relay node still responds to the first responder with an ACK packet. Although this Ack packet is sent in a unicast manner, the newly deployed relay is nevertheless able to receive this packet and on this way to determine the channel quality with its next relay in the direction of incident command. It's important that the relay has to make sure that it takes only packets from its direct neighbor relay into account, because, depending on the topology of the relays, it is possible that even a relay further away still responds over a very weak link.

4.3 Power Control

As we learned in section 3.4, the ChipCon CC1000 RF chip of the mica2 motes is able to change the transmitting power in very small stages. We want to take advantage of this in our dropping algorithm in two ways: First we want to add additional security margin to the dropping algorithm by starting to transmit not at full power so that we still have the possibility to increase our radio transmitting power to provide reliable connectivity. On the other side we also do not want to waste energy. Once the relay is dropped we can lower the transmission power so far that we have still a good radio link from one relay to the other but do not waste battery power. With lowering the transmission power we also reduce the possibility of collision between relays who could interfere with full power.

There are some criteria which have to be fulfilled that adaptive power adjustment can be done:

- The first responder itself never has to do power control, he always has to send with full power. If the first responder starts to adjusting its transmitting power the dropping algorithm with the channel probing is not reliable any more. Also because the first responder is most likely in continuous motion it does not make sense to adapt its transmitting power.

- As soon as a relay is in range of the first responder it has to set its transmission power to a preset start value. Because if the relay responds with different transmission power to the channel probe packets of the first responder the deployment algorithm would not work properly anymore

because stronger or weaker radio power would directly influence the place where another relay is dropped and that's not what we want.

- To allow power adjustment a routing algorithm with a routing table, which knows at least the next hop in direction first responder and in direction incident command, is necessary otherwise the relay does not know which links it has to adjust and which links are not important. More on this routing algorithm can be found in section 5.1.3.2

The algorithm itself works as follows: The relays have an array of well chosen discrete power values. These values were obtained from the measurement done in section 3.4. The relay starts in a power configuration which is two steps below maximum power and about 2.8 dBm below the maximum radio power. With every routing table message (more on this in section 5.1.3.2) the direct neighbor nodes not only advertise their routes but also advertise if the relay itself should lower, keep or even arise its power level. As soon as the relay realizes that it is out of range of the first responder or one of the neighbors wants a stronger link it rises the power level. If both routing neighbors (in our routing protocol we only have two important links, the one to the first responder and the one to the incident command) advertise to lower the power the relay goes to the next weaker power level.

The dynamic power adjustment produces no new overhead, because the power control values are included in the routing table packets. On the other hand power control brings the benefit of higher link reliability due to its additional margin in the radio link quality and helps to save battery power and minimizes interference between relays.

Chapter 5

Implementation

In this chapter we explain in more detail how the different parts of the prototype are implemented in software. First we focus on the Implementation on the mica2 motes, which is done in NesC. Furthermore we present the parts which run on the first responder IPAQ and the incident command laptop which are written in JAVA.

5.1 NesC

During implementing our prototype we always had in mind to produce well structured and reusable code. Therefore we tried to put as much code as possible in own, separated modules, which then can be used from the main program via simple wiring.

While implementing the dropping algorithm we decided to separate the TinyOS Software in two parts, the first responder and the relay software. The two nodes have totally different missions so it does not make sense to implement it in one program and always enter if clauses to determine if the software should act here as a relay or as a first responder. As mentioned above we tried to put software which is used by both parties, the relay and the first responder, in library modules which can be easily wired by both applications.

5.1.1 Transceiver

Because in our prototype every node and specially the first responder is sending many different messages in very short time intervals in an asynchronous way over the air but also over the serial interface, it was pretty soon clear that we need a module who coordinates and buffers the sending and receiving packets.

What makes the sending of packets in TinyOS not that easy has two reasons:

- First, for every type of message a new separate interface has to be wired and used. So sending different packet types at the same time can get confusing, because on the application layer you are using different sending interfaces but finally the end up interfering at latest in the MAC layer.
- TinyOS handles most tasks in an asynchronous way and gives back feedback over events. First if you try to send a packet you just get a response

if the packet was accepted or not. Later on you receive a successful sent event if the packet really was sent. If the packet was not accepted or not successful sent it's the task of the programmer to resend it. And furthermore the resending has to be done over a non-blocking task, otherwise the queue handling can be disturbed.

To summarize: it's extremely fault-prone to send many different packets type in short time in the traditional way. Also because you have to take care that you don't overwrite memory which still keeps important data.

Therefore we decided to write a library which centrally manages buffering and resending packets from all types of packets over the air and over the serial interface. The so called Transceiver library has one parameter which defines how large the TOS_Msg queuing pool of the Transceiver is.

To use the Transceiver in the implementation, wiring in the TinyOS Configuration file has to be done for every message type you use:

ModuelM.TransceiverAnyMsg-> TransceiverC.Transceiver[AM_ANYMSG];

The Transceiver then provides commands to get a TOS_Msg.data payload buffer and commands to send this buffer over the air or over the serial link. The Transceiver does not allow you to have more than one open buffer for the same message type at the same time. Obviously the Transceiver also provides events for receiving packets over the air and over the serial link. More details about the interface can be found in the Appendix.

The Transceiver module is used by the relays and the first responder. Having an own module which takes care of sending and buffering packets makes the other programming much more comfortable.

5.1.2 Dropping

The aim of this section is to show how the dropping algorithm, we discussed in chapter 4, is implemented on the node. The different role of the first responder and the relay in the algorithm reflects also in the two different programs for the relay and the first responder.

5.1.2.1 Relay

The implementation on the relays is pretty simple. The relay basically responds immediately to every broadcast channel probing packet it receives from the first responder. The first responder sends a *BcastMsg* which basically only holds his Id and a sequence number and the relay replays with a *RSSIMsg* in which the relay puts the same sequence number and its own Id.

The second task is the local placement aid we discussed in section 4.2. During the first minute when switched on the node also snoops packets not aimed to itself to measure the link quality to its next relay. Depending on the quality of the link, red or green Led indications are given.

As the base station is also a relay itself the base station relay has the special task to act as a bridge between air and serial interface. All packets which are addressed to the base station are forwarded to the serial link.

5.1.2.2 First Responder

The implementation of the first responder is considerably more complicated because it's the task of the first responder to find out where to place a new relay. Basically the first responder sends every 100 ms, if he is moving and every 500 ms, if he is not moving a *BcastMsg*, a broadcast channel probing packet. The first responder finds out if he's moving or not by checking its accelerator sensor values every 500 ms.

With every *RSSIMsg* packet which the first responder gets as the reply to its *BcastMsg*, the node starts applying the channel measurement algorithm: For every incoming packet the RSSI value is calculated, there is checked that there were no packets missed between the last received *RSSIMsg* and the time when the last packet for this node was arrived is updated (this time will be important later on). If no packets were missed the RSSI value is placed in the filter of the corresponding node and the sum of the RSSI values is updated. If packets were missed, additional default RSSI values are put into the filter. If everything is up to date a task, we give it the name CheckRSSI task, is started to check if the channel is still ok. Before we can explain this CheckRSSI task in detail we have to give some additional information. The first responder keeps track of which relay is the actual relay with the good radio channel and if at the moment we have a good link or we have a bad link and are waiting for another replay to replace the bad one. Only active relays are considered in the CheckRSSI task. A relay is then active when during the last two seconds at least one probe ACK message has arrived. To find out if a relay is active or not every 2 seconds a timer is checking this condition, therefore the updating of the last arrival time is important. During the short time needed to fill the filter with actual RSSI values after dropping a new relay or starting the first responder the CheckRSSI task also only considers the actual RSSI value and not the sum of the filter. In other situations the CheckRSSI task handles as follow:

- If we have an actual good link at the moment the task just checks if the sum of our actual link is still inside the channel quality limits. If this is true everything is still fine. If this actual good link has turned into a bad link the task searches for the best actual link among all active relays and tests if this one is over the threshold. If there is another link which is over the threshold this link becomes the actual good link and everything is fine. If there is no other link which is over the threshold the first responder switches to the kind of panic state where it has no link to a relay anywhere which fulfills our quality criteria. This also means that the first responder indicates over its LEDs that a relay has to be dropped instantly. To express this decision even more it's possible to switch on the beeper on the sensor board. But this sound becomes pretty soon very annoying.

- If the first responder is in its panic mode where it has no good link it checks if the last RSSI value which arrived in the form of a probing ACK packet is from a new relay and the RSSI value is over the threshold plus a small additional margin. If yes we are switching to an intermediate mode during a very short time which is needed to fill the buffer with actual RSSI value and during which we only consider the single RSSI value from the actual arriving packet. If the RSSI value is from the former good link

which turned into a bad link the task checks if the single RSSI value is over the threshold plus a small margin and the average of the filter itself is also over the threshold. If this criterion is fulfilled the task switches back to normal mode and switches back off the LEDs indication, otherwise it stays in the panic mode.

The code is quite complex because many timers are necessary and also overflowing of the data types, especially the time, has to be taken into account. Another important point is that single task should not be too much time consuming. Because there is no real threading that means single tasks are not preempted to fulfill other small tasks. Therefore it's mostly better to split on big assignments into smaller tasks otherwise the timing of these time crucial operations is getting unpredictable.

Additional to the channel probing packets the first responder tries to send its vital signs in a *VitalMsg* packet every few seconds to the base station.

Like the base station the first responder also has to forward communication packets which are addressed to the first responder to the serial port.

5.1.3 End-to-end Communication

The real aim of our prototype is still to provide reliable end-to-end communication between the first responder and the incident command. Till now we just discussed how we send messages to the next hop with the Transceiver module and how we find out where to place the next relay. But to provide the incident command to first responder communication we need another layer, the routing layer, which controls to whom messages have to be forwarded to reach its designated target.

Again we decided to implement this routing layer as separate library, which can easily be wired by both, the relays and the first responder. As a first approach we decided to use flooding because this is quite easy to implement and we needed a layer which is available as soon as possible for first tests. In a later version we then developed a true routing protocol specific to this prototype. Having a routing protocol is much more efficient in terms of usage of capacity of the network and also routing is the enabler for power control.

5.1.3.1 Broadcast

The broadcast module basically provides three different services: First, an unacknowledged send function to send messages in an unreliable way to the incident command and an end-to-end acknowledged reliable send function to the incident command and the first responder. We implemented the unreliable send function because the end-to-end acknowledges add a not negligible overhead to the protocol and if the transmission does not have to be 100% reliable it is more efficient to send them without ACKs.

To send a message over this broadcast library (all types of messages can be sent as long as they are not longer than 22 bytes) the original message is wrapped into a *BcastMsg* which has fields for a sequence number, the type of the message it holds, the destination, the sender and a flag if the messages has to be acknowledged or not. The combination of senderId and sequence number builds a unique identifier for every packet. To send an unacknowledged packet,

the broadcast module puts the unique identifier in a local memory and then broadcasts the *BcastMsg* with the included original message which has to be transported. If the module receives a *BcastMsg* it first checks if it has already received this specific packet. If yes, the packet is dropped. If it's a new packet the module checks if it's itself the addressee and if yes, provides the payload of the *BcastMsg* over the *Receive* interface to the upper layer. If it's not the addressee it just broadcasts forward the *BcastMsg*.

Sending the packets in a reliable, means acknowledged, way is a bit more complicated; although the mechanism is the same. As we mentioned above we implemented a very simple end-to-end acknowledge schema which allows only one single unacknowledged packet waiting for its ACK at one time. Because of this constraint of having only one open Acknowledge open, before sending a reliable packet the library checks if we are already waiting for an ACK or not. If not the packet is similarly sent as an unacknowledged but in addition we have to keep a copy of the payload packet in case of a necessary retransmission and the node also has to start a timeout timer in case the packet is lost and retransmission will become necessary.

The procedure of receiving a *BcastMsg* with the ACK flag set is the same as receiving an unacknowledged. Only if the node itself is the destination node a *BcastAckMsg* with the confirming sequence number has to be sent back. Because also this *BcastAckMsg* has to be broadcasted over many relays the *BcastAckMsg* is sent back with the same mechanism as all other messages using the broadcast protocol are sent.

If the original sender of the message, who waits for the acknowledgment, receives the *BcastAckMsg* with the correct sequence number the Acknowledgetimer is canceled, the Acknowledge-sending-lock is also canceled and the module checks if there are any queued messages to send.

If there arrives no corresponding *BcastAckMsg* in time at the sender, the Acknowledge-timer fires and the module resends the stored packet. Important in this case is, that the packet which is resent has to get a new unique sequence number otherwise the packet is not forwarded by the relays.

Customization of the broadcast library can mainly be done over the four parametes *BroadcastMemory*, *WaitForAckTime*, *NumberOfRetries* and *SendBufferSize*. BroadcastMemory defines how many already received unique sequence numbers should be kept in memory. The WaitForAckTime sets the time how long the module waits for an AckMsg before retransmission. The NumberOfRetries assigns how many times the module tries to resend a given packet before it finally drops it and SendBufferSize parameter defines how much memory is reserved to keep messages which have to wait before being sent. The three parameters WaitForAckTime, NumberOfRetries and SendBufferSize should always be changed together. Because it makes no sense to set the WaitForAckTime very high but the SendBufferSize very small, because that would mean that a lot of packets are not even been sent because they are discarded already while waiting. Neither is it good practice to set the BroadcastMemory and the SendBufferSize by default very high because as mentioned in section 2.2.1, memory is located at compile time and memory is a very rare commodity.

direction	nextHop	nrOfHops	badLinks	timeLastIn (μs)
Incident Command	9	3	1	24185815
Incident Command	8	2	2	26827403
First responder	10	4	0	24631349
First responder	11	5	0	25720490
First responder	7	3	1	24620105

Table 5.1: Example routing table for node with the Id four in figure 6.3.

5.1.3.2 Routing

Although the broadcasting module works fine for first rapid prototyping and in very small networks, it is very inefficient in larger networks. To get better performance we implemented an unicast routing protocol specially tailored to our prototype, which allows us to use link layer acknowledgements.

We decided to implement the routing module with the same interfaces as the broadcast module so that we need to change only one line of code in the Makefile to change between routing library and broadcast library. That means the module also provides commands to send reliable messages to the incident command and the first responder and additionally still allows sending unreliable messages to the incident command.

Route Discovery and Maintenance : The routing algorithm itself is a very specific adaptation of DSDV [1] to our prototype. A good paper addressing highly dynamic DSDV is [30]. The biggest differences are, that we do not use sequence numbers, link quality is part of the metric, and that we keep track of only two routes in our routing tables: the route to the incident command and the route to the first responder.

The algorithm keeps a table of routes in which the three best routes to the incident command and the three best routes to the first responder are stored. The route metric consists of the radio channel quality (in terms of RSSI) and the number of hops on the route. The best route is the one route which has the minimal amount of links with RSSI over a given threshold (bad links) and among these routes the one with the smallest hop count. Obviously the first responder and the incident commander have a special table because they always have itself as the only entry in one direction of the table.

The table entries consist of the Number of hops to the destination, the next hop on this route, the number of bad links the route contains till the destination and the time the last update for this route has come in. To make it a bit more concrete, an example routing table can be seen in table 5.1. This table corresponds to the routing table of node number four in the node topology, which is displayed in figure 6.3.

The nodes periodically advertise their best route in both directions to their neighbors by broadcasting a *RouteTableMsg*. This *RouteTableMsg* ist also sent if the node has a new best entry or if the best route in either direction changed its number of bad links on the route or in worst case if there is no route in any

[1] Destination-Sequenced Distance-Vector Routing (DSDV) is table-driven routing scheme for ad hoc mobile wireless networks based on the Bellman-Ford algorithm. Improved on the loop problem by using sequence numbers to mark each node.

direction anymore. There is also one special case: As soon as the first responder realizes that a new relay is dropped, it immediately advertises its routing table. This is necessary that the new relay gets as soon as possible the route from the fireman and triggered with getting this new routes also advertises its own table immediately to the other, older relays. This special advertisement of the first responder is triggered by the first responder program and not the routing library itself. The library just provides a command over which the table entries can be advertised.

Every time a *RouteTableMsg* arrives the routing tables entries are updated and if there's a new route, a new entry is inserted. When entering new entries care has to be taken to avoid loops. Therefore we do not enter routes in which the next but one hop is again us ourselves. When entering and updating the routing entries we also consider the RSSI value on the link to this neighbor to decide together with the information of the other node how many bad links the advertised route does contain. To have this RSSI information the library keeps a list of average RSSI values of all neighbors which is updated with every packet which arrives except the channel probing packets from the first responder because they arrive to frequently and also on this last link it's the business of the first responder to take care of the radio channel quality.

After some fixed time intervals the algorithm goes also through the tables to check that still the best route is in the first position (although no new *RouteTableMsg* are arrived the module can realize that a link to a neighbor turned into a bad link an therefore also the number of bad links on this route has to be adapted). At the same time the last update time is also checked. If there was no update over a given time period the routing entry is deleted.

Routing Data Packets : As mentioned above, the interfaces to the routing module are the same as the ones to the broadcast module. Identically to the broadcast module, the routing module wraps the message, which has to be sent, in its own packet.

Sending an unacknowledged packet is very similar to sending an unacknowledged packet in the broadcasting module. The only difference is that the packet is typically sent in an unicast manner to the next hop address, the routing protocol provides. The routing protocol looks up the next hop in the first routing entry (which is always the best) of the routing table. If for any reason there is no entry, which is very unlikely to happen, the packet is broadcasted instead of sending it via unicast. Also the forwarding process on the relay is almost the same as in the broadcast version; again with the difference that the message is not broadcasted while forwarded but sent to the next hop on the route to the destination.

Sending acknowledged packets though is very different from sending them in the broadcast module. First, the packet is wrapped in a *RoutingMsg*, which has the same fields as the *BcastMsg* from the broadcast protocol and again we first check if there is not already a package waiting for its ACK. We stayed with the decision to send the next reliable packet only after the preceding one has been acknowledged. This is no longer a big performance issue because of unicast routing we are now able to do hop-to-hop acknowledgement. Doing hop-to-hop acknowledgement results in a way shorter wait phase before receiving the ACK or resending the packet. If there is already a packet waiting for its ACK the

message is buffered as in the broadcast protocol. If there is no packet waiting for an ACK we look up the next hop from the routing table and send the package to this hop. We also keep a copy in case we have to resend the packet and we also store the sequence number we are expecting in the ACK packet. Although we send the packet in unicast manner it's still necessary to keep in memory the unique identifier because we can't be 100 % sure that that our neighbor will forward the packet also in unicast manner, because we have the broadcast backup mechanism. Again as we are sending reliably an AckTimeout timer is set.

If a relay receives a *RoutingMsg* it first checks if an ACK has to be sent. If yes it replies immediately with a *RoutingAckMsg* message which is sent in an unicast manner to the addressee. We do not need to send the *RoutingAckMsg* inside a *RoutingMsg* because this ACK is only a single hop ACK. We use a special *RoutingAckMsg* because this message is smaller than a basic *RoutingMsg* therefore the transmission is more efficient. Once the acknowledge is sent, the node checks if it is a new message (maybe just the ACK got lost or the message arrived via broadcast) and if yes the node forwards the message or if the packet reached already the destination the node delivers the message to the appropriate interface. Forwarding the reliable message is again the same procedure as sending it. If no (the packet is not new) just an ACK message is sent back.

If the sender receives the *RoutingAckMsg* within the AckTimeout time the timer is canceled, the sending lock suspended and the buffer is checked if other packets are waiting to be sent. If no *RoutingAckMsg* with the correct identifier arrives in time the AckTimeouttimer is fired and the packet is resent. It is not necessary to change the unique identifier. If the AckTimer is fired β times with the same routing entry, the entry is deleted from the table and the module tries again up to β times with the second and third entry in the table, if they exist. If there exists no other entry in the routing table, the protocol falls back to broadcasting with the difference that every node who receives the packet will acknowledge it. In case of broadcasting the sender will only wait for one ACK before proceeding to the next packet.

Like the broadcast module also the routing module can be customized by many parameters: As the broadcast module, the routing module has the three parameters *WaitForAckTime*, *NumberOfRetries* and *SendBufferSize*. As we do hop to hop ACKs with this module the *WaitForAckTime* can be much smaller whereas the other two parameters are subject to more or less the same criteria. There are additional three parameters to adjust the behavior of the routing algorithm timing: *SendRoutingtableTime*, *RoutingTableEntryTimeout* and *CheckRoutingTableRate*. The *SendRoutingtableRate* parameter defines how often the nodes advertise their table, the *RoutingTableEntryTimeout* table fixes after which time of no refreshment in the table the entry gets deleted and the *CheckRoutingTableRate* specifies how often the table itself is checked for entries who have timed out and correct order of the entries. More details how we have chosen the parameters can be found in the evaluation chapter in section 6.2.2.1.

5.1.3.3 Power Control

The mechanism of power control is more an add-on to the routing protocol module than a separate library; therefore we included it later on into the routing library. To do power control every node needs to know its next hop neighbors

to the first responder and the incident command. Additionally, we need information about the actual link quality. All information is already available in the standard routing library; the question is only how to use this information and how to inform the neighbors. But there is additional information we do not know yet: Is the first responder in radio transmission range or not? This is important, because as we presented in section 4.3, we are not allowed to change the transmission power if the first responder is in range. To gather this information we introduced another interface *IFiremanInRange* with two simple commands, *firemanInRange* and *firemanOutOfRange*, over which the relay node informs the routing library if the fireman is in range or not. The relay application can do this much better because only this application cares about the channel probing packets from the first responder.

To enable power control we added two fields in the *RouteTableMsg*. Remember in *RouteTableMsg* messages the nodes advertise their best route to the first responder and also the best route to the incident command. We now add to both routes if we like to have more or less power on these links or if the received power on the link is just fine. The receiving relay then has to check if he is the next hop on the advertised route and if yes has to take into account the power control request of its other neighbor. Then if additionally the first responder is not in range and both neighbors are requesting less power, we drop our power; if at least one neighbor wants more power, we raise the power. As mentioned already in section 4.3 the library has an array with discrete power values to choose from. The exact values for the power levels can be found in section 6.3.1.

5.1.4 Debugging

Normally debugging is not really an implementation issue. But to debug a distributed network with sensor nodes with no display, debugging can become a challenging thing. Within a sensor network there are basically four different methods available to debug:

- **LEDs** - For simple debugging the three color LEDs are enough. Indicating simple events or states with the LEDs is an easy and fast way to debug. But for more complicated debugging three LEDs are obviously not enough.
- **Debug messages over the air** - Is a relatively easy way to get data from the motes to the computer. Has the big disadvantage that you need a working multihop environment, it takes away network bandwidth and it's not reliable due to collisions.
- **Debug messages over the serial interface** - Is a relatively simple way to transfer data from the mote to the computer. Has the big disadvantage that it's only feasible for the base station which is over a gateway connected to the computer.
- **Logging onto the EEPROM** [2] of the mote and later on transfer it over the air or serial interface to another computer - this is a very convenient way to perform reliable logging directly on the mote. The disadvantage is, that the logging is slower than sending messages.

[2] Electrically Erasable Programmable Read-Only Memory, is a non-volatile storage chip used in computers and other devices to store small amounts of volatile (configuration) data.

We implemented a simple logging module which is very easy to wire to any module and quite easy to adapt to every kind of data type you want to save. At the end we transfer the data to the computer where we save the data into a file, a database or only display it in a console.

Additional to this logging module we implemented directly into the broadcast and the routing library a simple mechanism to report the connectivity of every relay to each other. We implemented this mechanism directly into the library because we wanted to have this information in real time and for that we use directly the unreliable sending interface of this library. We implemented this real time connectivity report because we realized that this is almost the only way to find out if the deployment algorithm works fine and what kind of topology it produces.

5.2 JAVA

In this section we present the software we implement in Java for use on the client side, mostly GUIs to display data and to enable communication. We decided to display the different GUIs in its own window so that we can flexibly decide which windows we want to display.

As mentioned in section 2.2.2 we use an adapted Serialforwarder which comes already with TinyOS to translate the TinyOS packets from the serial interface to the Java program.

5.2.1 Communication GUI

The Communication GUI which is displayed in figure 5.1 is used to allow text messaging between the incident command and the first responder.

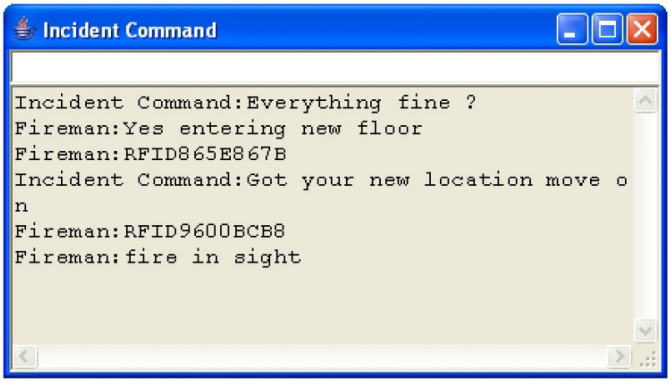

Figure 5.1: Text messaging communication GUI on the incident commander side - the one on the IPAQ looks identical.

In figure 5.1 the incident command side window is showed; the same window appears on the first responder side on the IPAQ. This is also the only GUI which is running on the IPAQ; the display is just too small to show more information. The Communication GUI is basically a very simple instant messenger as there

exist nowadays many on the internet. It is also the responsibility of the communication GUI to split long messages into smaller packets and again build the messages out of many small packets on the other side. This fragmentation is necessary because the payload of our *RoutingMsgs* is only 22 byte. The text field on the top is used to enter a new message, hitting enter sends it to the other party. In the big pane below the messages from both contributors are displayed.

5.2.2 Vital Signs GUI

The Vital signs GUI which is displayed in figure 5.2 has the duty to display the evolution of the sensor values of the first responder. In real life this would contain additional vital signs like heart rate and so on. To be able to display the evolution of the data, the vital sign GUI has to maintain a data collection to save the received values.

Figure 5.2: Vital signs GUI to display the sensor readings of the first responder.

5.2.3 Topology GUI

The topology GUI displays the logical topology together with the connectivity of the relays and is produced out of the debug information we discussed in section 5.1.4. In figure 5.3 we can see a topology of eight relays together with the first responder on the right side. The topology is the easiest way to verify how accurate our deployment algorithm works. The labels on the links indicate the actual RSSI values of the link in dBm. The numbers in the yellow circles correspond to the node Ids of the relays.

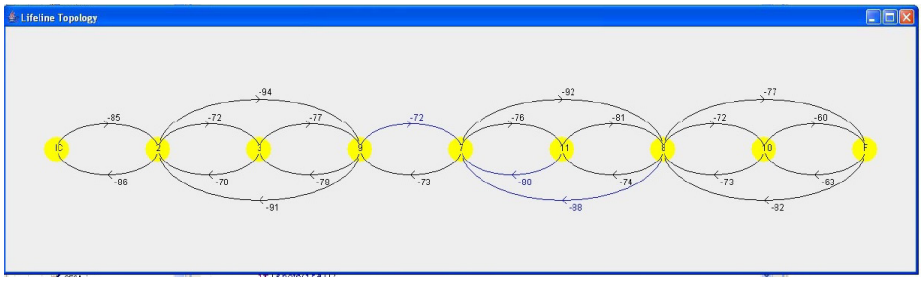

Figure 5.3: Topology GUI which displays the topology of the relays (the numbers are just Ids of the relays) - the link labels are in dBm.

5.2.4 RFID GUI

The RFID GUI displays the actual position of the first responder based on the RFID tag values the first responder is reading with its RFID reader plugged into its IPAQ. In figure 5.4 you can see how the GUI looks. At the moment every RFID tag ID is linked to a specific position in the building. In a later real world application the idea would be to have 3-D coordinates on the RFID tags itself and the GUI then could display the actual floor plan out of the coordinates if the floor plans would be digitalized in advance and saved in a database which should be accessible by the incident command.

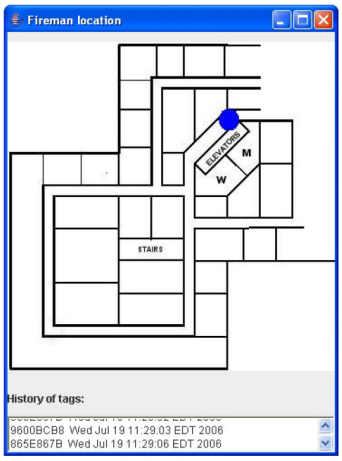

Figure 5.4: Location GUI to display the location of the first responder accordingly to the RFID tag he crossed.

Chapter 6

Evaluation

To evaluate our dropping algorithm and also the reliable text messaging with the broadcast and the routing library we tested in a real environment. As our test environment we used the building where our office is located. As the start point we always used my own office. A detailed floor plan can be seen in figure 6.1 As

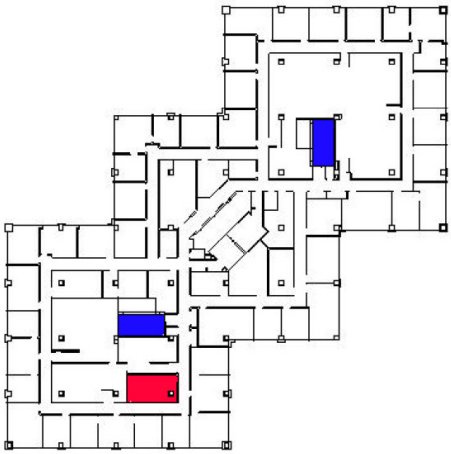

Figure 6.1: Floor plan of fourth floor NIST north building - my office in red, stairwells in blue.

one can see in the floor plan the building allows many different and interesting ways to test the prototype. Our main goal was to test the path all the way down to the entrance of the building, because this would be the typical application of the system: An incident command is established outside the building and the first responder is entering the building over stairs in the stairwell.

One big problem we encountered during our tests was the impact of other working people around the building. Not that they would disturb the radio propagation, it was more the curiosity and safety awareness of other employees. Because the sensor nodes we placed as relays on the floor look quite suspicious, we often found our mica2 motes all the way down at the security office desk.

Even after furnishing our relays with notes that there's a test in progress people picked up the relays and placed them back down at different places. So the solution was to perform most of our test during late hours or weekends when no other employees were around.

6.1 Dropping the Relays

To evaluate the performance of the dropping algorithm and to find suitable parameters as discussed in section 4.1.3 we did a great deal of test runs to drop the relays on many different routes. The resulting topologies of some of these runs are shown in figure 6.2 and 6.3 as well as in figure 5.3 in chapter 5. All three topologies were produced with local placement adjustment aid and a threshold of -80 dBm.

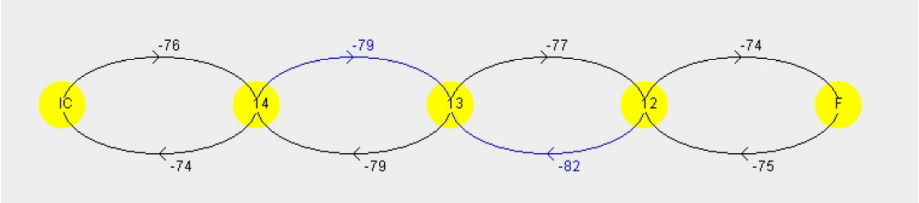

Figure 6.2: Evaluation run with four relays on the same floor.

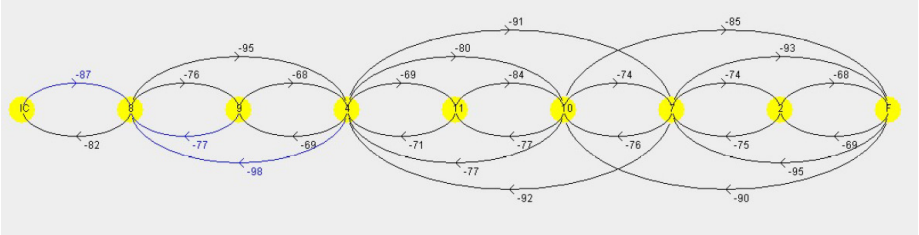

Figure 6.3: Evaluation run with eigth relays crossing three floors, stairwells and two closed doors.

A very basic evaluation of the dropping algorithm is very simple: If at the end of the dropping we still have connectivity between the first responder and the incident command the dropping was successful. Including the help of the local placement mechanism of the relays, as discussed in section 4.2, we achieved a success rate of 100%. (We know that stating 100% success rate is very big-headed. But in our recent tests we had no failures). 100% success rate means at the end all produced topologies had connectivity between the first responder and the indicent command.

Besides the simple evaluation - connection or not - it is more interesting also to evaluate how close the link quality on links to immediate neighbors stays to the aimed threshold. In the recent runs all links stayed inside the range of -68 and -87 dBm. That means the links are at most 7 dBm too weak and 12 dBm

too strong. Links which are too strong normally are not a problem. And Given that we have a margin of about 15 dBm (remember our aimed level is -80 dBm and the critical level is -95 dBm), having links with -87 dBm as the biggest difference from the threshold is very satisfying.

Having 100% reliability as an objective it's comforting to see that most links are even stronger than the aimed -80 dBm. On the other hand one can argue, that too strong links results in shorter distance between relays.

As a summary we can say we are very pleased with the result we obtained using our deployment algorithm.

6.2 Communication

To test the efficiency of the broadcast and the routing module over the deployed ad-hoc network we did the following tests: During the entire duration of the test, that means already during the deployment of the relays, we were sending two types of messages:

- Unreliable, means unacknowledged, vital sign messages from the first responder to the incident command.

- Reliable, means acknowledged, text messages which where sent roundtrip from the incident command to the first responder and back. The Java software on the IPAQ replied to these messages, so the round trip was at the application layer.

For both message types we measured the packet loss rate whereas a packet loss rate other than 0% for the reliable messages would mean a failure of the protocol.

6.2.1 Broadcast

We tested the broadcast library over many different topologies. Recall that "broadcast" messages are flooded on the network and that reliable messaging is implemented with end-to-end acknowledgment and retransmission in this case. While sending the reliable text messages every five seconds we did not miss any single roundtrip packet. Unreliable vital sign messages arrived with an average packet loss rate of 13%. These results are taken over 500 reliable and 500 unreliable packets. Looking at the traffic the broadcast module generates, we realized that even with a small rate of one packet every five seconds there were retransmissions on the broadcast library layer on almost every second packet.

Because it is quite obvious that broadcasting with end-to-end acknowledgment is not very efficient we did not put too much effort into exploring the maximum sending rate nor in finding the optimal parameters but rather invested time into developing the routing module.

Nevertheless, as a first result we were pleased to achieve 100% reliability with our simple broadcast module even though a packet rate of one packet every five seconds is very low.

parameter	value
BroadcastMemory	5
WaitForAckTime	1000 ms
NumberOfRetries	5
SendBufferSize	20

Table 6.1: Parameters used for the broadcast module in our tests.

timeout	packets sent	packets resent	% resent
150 ms	1125	106	9.4
125 ms	1058	86	8.1
100 ms	1042	510	48.9

Table 6.2: Results of the measurement with different timeout times for the acknowledgement.

6.2.1.1 Parameter

Here we want to present the values we have chosen for the parameters presented in section 5.1.3.1. The values can be found in table 6.1.

What is particularly to point out is, that we had to set the timeout for acknowledgements up to one second otherwise, we had much too much unnecessary retransmission. The problem there is that the processing time of the broadcast packet in every node is not negligible. At every node the forwarding of the packet takes some time. This processing time is very unpredictable because it depends on the load in the task queue. So the big delay results not because of transmission time and time the signal is in the air, it's mostly the processing time on the nodes.

6.2.2 Routing

We also tested the routing library with different topologies. We achieved with the routing library 100% reliability for the acknowledged messages with a packet send rate of one packet per second. But because the routing library with its one hop based acknowledgement should be much more efficient in theory we wanted to find out the maximum packet send rate which still allows 100% reliability. To find out this maximal send rate we first had to find out an adapted value for the timeout. After some experiments with different timeout values, the results can be seen in table 6.2, we found out that 125 milliseconds is a good tradeoff between waiting too long and having too much unnecessary retransmission due to early retranmissions.

With this timeout of 125 ms we achieved a packet transmission rate of the roundtrip text message packets of one packet every 350 ms still keeping the 100% reliability. This test was done over a three hops topology with over 500 packets. The number of hops should not play a relevant factor because every hop acts again as an autonomous end-to-end acknowledgement segment.

Amazingly the packet loss rate in such a highly loaded network for the unreliable vital sign messages was again 13%. It does admittedly not make sense to still send some packets over the unreliable send interface because the overhead for the acknowledgement is acceptable but we were however interested in the

parameter	value
BroadcastMemory	5
WaitForAckTime	125 ms
NumberOfRetries	15
SendBufferSize	20

Table 6.3: Basic parameters used for the routing module in our tests.

parameter	value (ms)
SendRoutingTableRate	1500
RoutingTableEntryTimeout	3000
CheckRoutingTableRate	7000

Table 6.4: Routing specific values we used for the routing module in our tests.

result.

We can see it as a success that we achieved 100% roundtrip reliability with a packet rate of one packet every 350 ms. On the other side three packets a second is not a very exhilarating packet rate. One of the big problems seems to be that we lose much time in processing the packet at each node. A 125 millisecond timeout time is still a long time. It might also be that our adapted MAC protocol slows the throughput down. In worst case the roundtrip initial backoff can sum up to 106 ms. Together we the bytes from the packet itself the worst case roundtrip transmission time can sum up to 143 ms. A more enhanced sending protocol, which would use a kind of sliding window, could probably enhance packet rate.

6.2.2.1 Parameter

Table 6.3 lists the values for the parameters we used for the routing module. The basic parameters *BroadcastMemory* and the *SendBufferSize* are the same as in the Broadcast whereas the *WaitForAckTime* is much smaller, as discussed above. We also set the *NumberOfRetries* up to 15. First We do lose relevant less time between two retransmission therefore we have to arise the value that in case of a short out of range situation of the first responder we try long enough.

The routing specific values can be seen in table 6.4. To emphasize is, that the parameter *SendRoutingtableRate* is with 1.5 seconds quite small. We have to set the retransmission that small because our protocol is based on the fact, that even close to the first responder, where the topology changes quite quickly, the tables are up do date. We also have to say that we did not have much time to find out the best parameters for the routing particularly because there are still open points as you will se in the outlook section 7.2.

6.3 Power Control

In this section we want to evaluate the impact of the power adaptation mechanism to the network topology, the dropping algorithm produces. For this reason we walked along the same path twice, once with power adaptation on and once without power adaptation. The results can be seen in figure 6.4 and figure 6.5.

Comparing the two figures what catches one's eye the most is that with the power adapt mechanism on we needed seven relays and with power adaptation off only six relays. This is most likely because with the power adaptation off the relays are always sending at full power whereas with the more adaptive power control mode the motes start with a power level which is -3.83 dBm below the maximum level and they only increase power if necessary. Therefore if the dropping algorithm worked perfect, all relays in the power adaptation mode are sending with -3.83 dBm below the maximum.

Comparing the link qualities between the two networks we notice on the unadapted topology, as already stated in section 6.1, variations from the aimed threshold in a range of twelve dBm too strong and five dBm too weak. On the adapted graph the variations are as expected much smaller. They range from one dBm too weak (We do not count the link with -83 dBm to the first responder, because remember links which are in range of the first responder do not do power adaptation) and six dBm to strong. This is a much smaller range and specially that the worst link is only one dBm over the threshold shows that the power adaptation works very well.

In terms of how many links to next neighbors they have, both topologies are almost the same. But most nodes in the power adapted version still have the possibility to increase the power at least one level therefore the links to the next but ones could still be improved in case of a failure of the direct neighbor.

In practice the question arises if one is up to place a few more relays to gain more security margin and more accuracy on the radio links. To be able to make more detailed conclusion much more tests have to be done for which we unfortunately did not have the time yet.

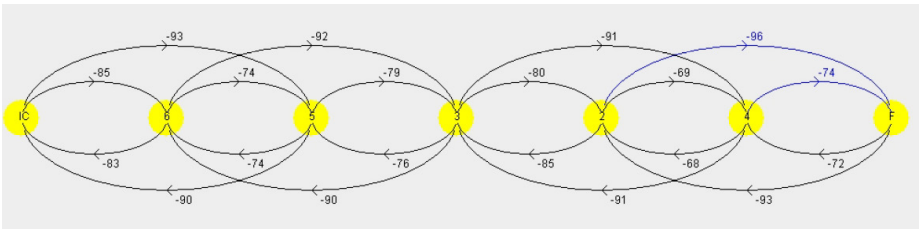

Figure 6.4: Evaluation run with six relays on the same floor **without** power control.

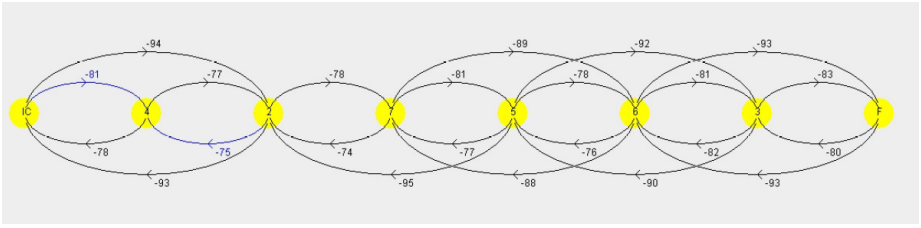

Figure 6.5: Same path as above but **with** power control and seven relays.

power level	value	Δ dBm
12	255	0
11	239	-2.41
10	**191**	**-2.79**
9	175	-3.83
8	159	-4.72
7	143	-5.63
6	127	-6.73
5	95	-8.48
4	78	-9.46
3	63	-10.19
2	15	-13.89
1	9	-17.93

Table 6.5: Power levels used in the power adaptation algorithm - default power level is 10.

6.3.1 Parameter

The only parameters we used for power control is how close to the aimed threshold we want to come and the power levels itself. We have chosen to target directly the value of -80 dBm without any margin. The second parameters we had to choose are the power levels. We decided to chose 12 different power levels which can be seen in table 6.5. The start value and also the value the nodes switch to, as soon as the first responder is in range is power level ten. The values are obtained using the results presented in section 3.4. For a future implementation it would possibly make sense to choose more power levels with smaller distances between each other.

Chapter 7

Conclusion

7.1 Result

In this thesis we developed a prototype implementation of an ad-hoc based communication environment where the relays are channel probing based in real time deployed. The final product with its three parts: first responder, relays and incident command can be seen all together in figure 7.1.

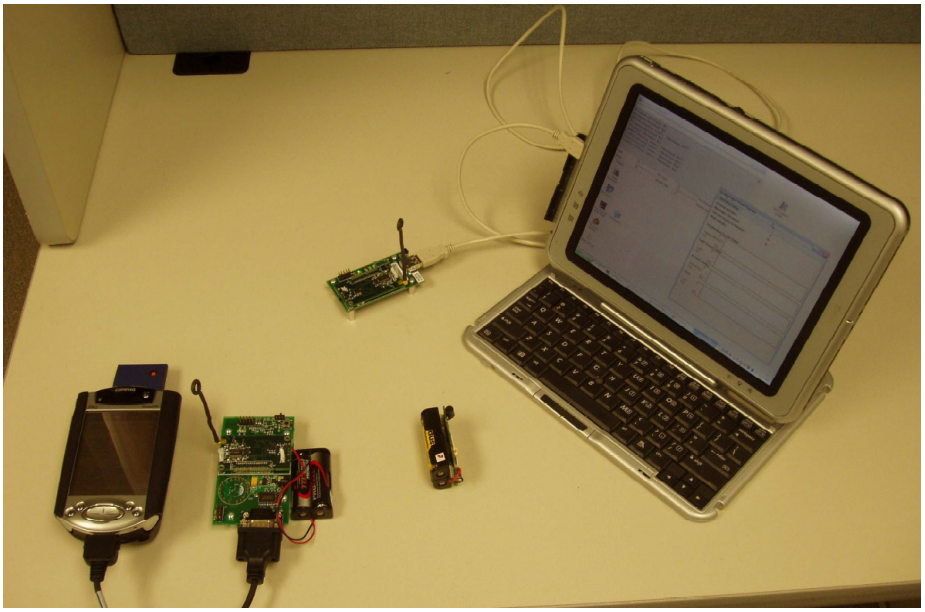

Figure 7.1: Overview over the working prototype: first responder node with attached IPAQ which serves as the display and RFID reader (left), a single relay with the small embedded antenna (middle), and the incident command laptop with its attached relay (right).

Although it was not clear at the beginning if the mica2 sensor motes are capable of mastering this task, experiments and evaluation as presented in chapter

6 have showed that our algorithm works and we can use the produced network for reliable communication.

But there is still a lot of future work to do. We point out the most important future tasks in the next section.

7.2 Outlook

In the previous chapters we presented a first working basic prototype. There are still many open points to solve and things to enhance. During the work on this project many ideas for extensions were proposed. Many of them found their way already into the prototype others had to be postponed or given up due to time or hardware constraints. We want to show in the following list where future work will be necessary.

- To bring this prototype closer to a real world suitable implementation a more powerful and more adapted hardware platform has to be evaluated and used. More powerful in the ways of a wider radio range and also in the way of having higher bandwidth. This higher bandwidth then would also be suitable to provide real-time voice and even video services which are both elementary in first responder scenarios.

- At the moment the prototype supports only one single first responder entering a building. In real cases many first responders are entering a building therefore the prototype has to be adapted to support more than one first responder. This means also to adapt our routing algorithm to handle more than two routing destinations.

- The routing algorithm still offers a range of enhancements. Specially loop prevention is still very basic and hast to be improved. In worst case situations there is still the possibility that our routing algorithm produces loops. Another adaptation which could improve the performance is to separate the network in a fast changing part in the surrounding of the first responder and a more stable part. That would mean we could adapt the rate to send the routing tables concerning in which are each node stays. Separating the network in a more stable and a dynamic part could also be the base to adapt the MAC layer parameters, mainly the backoff, dynamically to the situation. The routing algorithm could also take advantage of the channel probing packets to update the table more rapidly in the fast changing part of the network.

- On the basis of the acknowledgement protocol it would be nice and more efficient to have a sliding window mechanism then waiting for the acknowledgement for every packet before sending the next one.

- To make the dropping algorithm even more reliable it would be worth to try a dropping schema in which the first responder always maintains two good radio channels to two different relays.

- It would be nice to make the link quality metric even more reliable and not only rely on averaging RSSI measurement. It would be very useful to find a good and robust method to avoid placement in deep fading. A possible

approach could be to take also the speed of the alteration of the RSSI signal into account. Or maybe even using a dual antenna and include more details like the direction of the signal and or phase shift into the algorithm.

- Regarding TinyOS itself it will make sense in medium term to switch over to TinyOS version 2 which seems to allow more enhanced task scheduling which again would allow to handle more important tasks first. Also the complied code itself seems to be more efficient in version 2. Another important task is to switch to dynamic memory allocation. At the moment the first responder node uses almost all its four kbytes of SRAM.

- In our prototype we introduced very basic power control. We think that in this field we can add even more reliability and robustness with a more sophisticated algorithm to handle the transmission power more.

- As a last open point much more tests in different indoor environments have to be made. We have seen that signal propagation inside buildings is very hard to forecast. We did all our tests in the same building so it will be interesting to evaluate the prototype in a different building with different walls and different floor plans.

7.3 Personal Experience

The work on this master thesis was very exciting during all its stages. It was interesting to work the first and probably also the last time with a small embedded platforms where the constraints given through the limited resources of the sensors impose different approaches to solutions than on normal computers, computer scientists normally uses. It's fascinating to see that many of these small distributed sensors with limited resources can build a very strong and efficient system if they work together.

It was the first time I also really encountered the kind of research directed work. At the beginning it was not very clear if that, what we have in mind to do, is feasible with the platform we have chosen. So it was always a challenge to achieve the results we were looking for.

It was also very interesting to work with a very broad range of hardware. From soldering antennas to PC boards, to configuring PDA's over to programming in nesC, Java and even using databases, this thesis included a very broad variety of technologies which made it very interesting.

Another very enriching part was the change to write this thesis in a research lab in the United States. It was very interesting to encounter how research is done in a different institute than ETH Zürich.

Of course there were also the less enjoyable sides of this project. First TinyOS is still involved in a continuous change. Therefore things often are not very well documented and change from one version to the other without any announcement. Another very annoying thing is to debug the sensor nodes. Because they have no display and there exists almost no tools to support the developer it can be quite frustrating to find bugs in the nesC code especially for computer scientists who are used to have the luxury of an Eclipse debugging environment. Besides debugging, the hardware itself can be the reason for

perplexity, specially again for computer scientist who are used to do everything in software. You can never be quite sure if a malfunction is based on hardware or software misbehavior. Small things like a weak battery or a lose contact to the antenna can be the cause for unexpected behavior but in some cases the reason for surprises even remains unknown, especially when the motes switch back to normal operation without any influence. Nevertheless this was a great choice for my master thesis and I will keep it in excellent memory as a challenging experience.

Appendix A

Interfaces and Messages

In this chapter we want to present the TinyOS interfaces our modules provides.
Furthermore we show in detail the messages the modules use.

A.1 Interfaces

A.1.1 Transceiver

```
interface ITransceiver {
/**
   * Request a pointer to an empty TOS_Msg.data payload buffer.
   * This will allocate one TOS_Msg to the requesting AM type until it is sent
   *
   * You must call sendRadio(..) or sendUart(..) when finished
   * to release the pointer and send the message.
   *
   * @return a TOS_MsgPtr to an allocated TOS_Msg if available,
   *         NULL if no buffer is available.
   */
  command TOS_MsgPtr requestWrite();

  /**
   * Release and send the current contents of the payload buffer over
   * the radio to the given address, with the given payload size.
   * @param dest - the destination address
   * @param size - the size of the structure inside the TOS_Msg payload.
   * @return SUCCESS if the buffer will be sent. FAIL if no buffer
   *         had been allocated by requestWrite().
   */
  command result_t sendRadio(uint16_t dest, uint8_t payloadSize);

  /**
   * Release and send the current contents of the payload buffer over
   * UART with the given payload size.  No address is needed.
   * @param size - the size of the structure inside the TOS_Msg payload.
   * @return SUCCESS if the buffer will be sent. FAIL if no buffer
```

```
   *         had been allocated by requestWrite().
   */
  command result_t sendUart(uint8_t payloadSize);
  /**
   * A message was sent over radio.
   * @param m - a pointer to the sent message, valid for the duration of the
   *     event.
   * @param result - SUCCESS or FAIL.
   */
  event result_t radioSendDone(TOS_MsgPtr m, result_t result);

  /**
   * A message was sent over UART.
   * @param m - a pointer to the sent message, valid for the duration of the
   *     event.
   * @param result - SUCCESS or FAIL.
   */
  event result_t uartSendDone(TOS_MsgPtr m, result_t result);

  /**
   * Received a message over the radio
   * @param m - the receive message, valid for the duration of the
   *     event.
   */
  event TOS_MsgPtr receiveRadio(TOS_MsgPtr m);

  /**
   * Received a message over UART
   * @param m - the receive message, valid for the duration of the
   *     event.
   */
  event TOS_MsgPtr receiveUart(TOS_MsgPtr m);
}
```

A.1.2 Broadcast

provides the standard interface `Receive[uint8_t id]` to receive messages and `Send[uint8_t id]` to send unreliable messages to the indident command. Additionally:

```
interface IReliableSendEndtoEnd {
  /**
   * Send a message buffer with a data payload of a specific length in a reliable
   * way to the base station or the fireman.
   * The buffer should have its protocol fields set already, either through
   * a protocol-aware component or by getBuffer().
   *
   * @param msg The buffer to send.
   * @param length The length of the data buffer sent using this
   * component. This must be <= the maximum length provided by
```

```
   * getBuffer().
   *
   * @return Whether the send request was successful: SUCCESS means a
   * sendDone() event will be signaled later, FAIL means one will not.
   */

   command result_t sendBase(TOS_MsgPtr msg, uint16_t length);
   command result_t sendFireman(TOS_MsgPtr msg, uint16_t length);

  /**
   * Given a TinyOS message buffer, provide a pointer to the data
   * buffer within it that an application can use as well as its
   * length. If a protocol-unaware application is sending a packet
   * with this interface, it must first call getBuffer() to get a
   * pointer to the valid data region. This allows the application to
   * send a specific buffer while not requiring knowledge of the
   * packet structure. When getBuffer() is called, protocol fields
   * should be set to note that this packet requires those fields to
   * be later filled in properly. Protocol-aware components (such as a
   * routing layer that use this interface to send) should not use
   * getBuffer(); they can have their own separate calls for getting
   * the buffer.
   *
   * @param msg The message to get the data region of.
   * @param length Pointer to a field to store the length of the data region.
   * @return A pointer to the data region.
   */

  command void* getBuffer(TOS_MsgPtr msg, uint16_t* length);

  /**
   * Signaled when a packet sent with sendBase() or sendFireman() completes.
   *
   * @param msg The message sent.
   * @param success Whether the send was successful.
   * @return Should always return SUCCESS.
   */
  event result_t sendDone(TOS_MsgPtr msg, result_t success);

}
```

A.1.3 Routing

Provides the same interfaces as presented in Broadcast A.1.2 plus additionally:

```
interface IRouting {
   /**
   * Broadcasts the best routingTable entries in both direction, to the
   * indident command and the first responder
   *
```

```
   * @return SUCCESS if the rouintTable entries have been broadcasted. Fail if
   * sending failed.
   */
   command result_t broadcastTable();

   /**
   * Updates the RSSI average of the specific node with the RSSI value
   * delivered with the command
   *
   * @param from - the node address from which is the RSSI value
   * @param RSSI - the RSSI value measured on the link to the node
   * @return SUCCESS if the average has been updated. FAIL if updating
   * failed.
   */
command result_t updateRSSI(uint8_t from, int16_t RSSI);
}

interface IFiremanInRange {
   /**
   * Indicates to the routing module that the first responder is in radio range.
   *
   * @return SUCCESS if change is noticed. Fail otherwise
   */
   command result_t firemanInRange();

  /**
   * Indicates to the routing module that the first responder is out of radio range.
   *
   * @return SUCCESS if change is noticed. Fail otherwise
   */
   command result_t firemanOutOfRange();
}
```

A.2 Messages

A.2.1 First Responder and Relay

```
//Message to send vital signs from the first responder to the incident command
typedef struct VitalMsg{   //13 bytes
uint8_t  from;    //from address
uint16_t nr;    //sequence nr
uint16_t voltage;    //voltage of the node
int16_t  rssi;    // average RSSI of best link
uint16_t temp;    //temp value
uint16_t light;    //light value
uint16_t accel;   //absolut accel value
} VitalMsg;
```

```
//message to probe the channel
typedef struct BcastMsg{
uint8_t from; //from address
uint16_t nr;        //sequence nr
} BcastMsg;

//message to answer to the probes
typedef struct RSSIMsg{
uint8_t from;   //from address
uint16_t nr;    //sequence nr
} RSSIMsg;

//message to transfer data between first responder and incident command
typedef struct payloadMsg {
  uint8_t startId;
  uint8_t id;
  uint8_t length;
  uint8_t payload[18];
} payloadMsg;
```

A.2.2 Broadcast

```
//Broadcast Header message to wrap payload into
typedef struct BcastMsg {
  uint16_t seqno; //sequence Nr
  uint8_t amType;   // am Type of payload
  uint8_t from; // from
  uint8_t to;       // to
  uint8_t lastHop;  // las hop on way
  uint8_t ACK;      // ack yes or no
  uint8_t data[(TOSH_DATA_LENGTH - TOSH_HBCAST_HEADER_LENGTH)]; //22 bytes payload
}BcastMsg;

//Broadcast Ack message
typedef struct BcastAckMsg{
uint16_t seqno;  //sequence Nr
uint8_t to;      //to
uint8_t from;  //from
} BcastAckMsg;

//Debug message to display toplogy and connectivity
typedef struct NeighborMsg{
uint8_t from; //from
uint8_t nrOfValidEntries;  //nr of Neighbor entrie
NeighborEntry  entries[6]; //20 bytes
}NeighborMsg;

typedef struct NeighborEntry { //3 byte
  uint8_t nr;
  int16_t RSSI;
```

```
} NeighborEntry;
```

A.2.3 Routing

```
//Routing table advertisment
typedef struct RouteTableMsg {
  uint16_t seqno;  //sequence Nr
  uint8_t from; //from
  RoutTableMsgEntry entries[2];   //first fireman second base
  int8_t trendFireman; //power control Fireman
  int8_t trendBase;     //power control base
} RouteTableMsg;

//Entry in RoutingTableMsg
typedef struct RoutTableMsgEntry {
  uint8_t to;  //to whom
  uint8_t nrOfHops;   //nr of hops
  uint8_t via; //first hop on route
  uint8_t badlinks;   //number of bad links on the route
} RoutTableMsgEntry;

//Routing message to wrap payload into
typedef struct RoutingMsg {
  uint16_t seqno;  //sequence Nr
  uint8_t amType;  //am Type of paylad
  uint8_t from;  //from last hop
  uint8_t fromOrig;  //from originaly
  uint8_t to;   //to
  uint8_t ACK;    //Ack yes/no
  uint8_t data[(TOSH_DATA_LENGTH - TOSH_HROUTING_HEADER_LENGTH)]; //22 bytes payload
}RoutingMsg;

//Routing Ack message
typedef struct RoutingAckMsg{
int32_t seqno;  //sequence Nr
uint8_t to; //to
uint8_t from; //from
} RoutingAckMsg;

//Debug message to display toplogy and connectivity
typedef struct NeighborMsg{
uint8_t from; //from
uint8_t nrOfValidEntries;  //nr of Neighbor entrie
NeighborEntry  entries[6]; //20 bytes
}NeighborMsg;

typedef struct NeighborEntry { //3 byte
  uint8_t nr;
  int16_t RSSI;
} NeighborEntry;
```

List of Figures

List of Tables

References

[1] SpearnNet Team Member Radio.

http://www.acd.itt.com/pdf/SpearNet_10-04.pdf

[2] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler. The emergence of networking abstractions and techniques in tinyos. In Proceedings of the First USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI), 2004.

http://db.lcs.mit.edu/madden/html/tinyos-nsdi04.pdf

[3] Aditya Mohan, Wei Hongt, David Gayt, Phil Buonadonnat, Alan Mainwaringt. End-to-End Performance Characterization of Sensornet Multi-hop Routing. In IEEE ICPS, 2005.

http://ieeexplore.ieee.org/iel5/10064/32279/01506386.pdf?isnumber=&arnumber=1506386

[4] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In Proceedings of the First International Conferenceon Embedded Networked Sensor Systems (SenSys), 2003.

http://www.cs.berkeley.edu/~awoo/sensys_awoo03.pdf

[5] Konrad Lorincz and Matt Welsh. MoteTrack: A Robust, Decentralized Approach to RF-Based Location Tracking. Personal and Ubiquitous Computing, Special Issue on Location and Context-Awareness, Springer-Verlag, 2006. In press.

http://www.eecs.harvard.edu/~mdw/papers/motetrack-loca05.pdf

[6] Tereus Scott. Mica Mote Antenna Radiation Pattern Analysis.

http://www.cs.uvic.ca/~wkui/research/motereport-p.pdf

[7] Nana Dankwa. An Evaluation of Transmit Power Levels for Node Localization on the Mica2 Sensor Node.

http://groups.csail.mit.edu/drl/journal_club/papers/nana.dankwa.ee.pdf

[8] Martin Kubisch, Holger Karl, Adam Wolisz, Lizhi Charlie Zhong, Jan Rabaey. Distributed Algorithms for Transmission Power Control in Wireless Sensor Networks. In Proc. of IEEE Wireless Communications and Networking Conference (WCNC), New Orleans, Lousiana,USA, March 2003 IEEE.

http://www.tkn.tu-berlin.de/publications/papers/kubischTxCtrl_accepted.pdf

[9] Fortune, S. J., Gay, D. M., Kernighan, B. W., Landron, O., Valenzuela, R. A.,and Wright, M. H. WiSE Design of Indoor Wireless Systems: Practical Computation and Optimization. IEEE Comput. Sci. Eng. 2, 1 (1995).

http://ieeexplore.ieee.org/iel4/99/8543/00372944.pdf?tp=&arnumber=372944&isnumber

[10] Helicomm Mesh Network modules.

http://www.helicomm.com/download/8051%20Module%20Datasheet_0406C.pdf

[11] Crossbow Technology Inc.

http://www.xbow.com

[12] ChipCon CC1000 Datasheet.

http://www.chipcon.com/files/CC1000_Data_Sheet_2_2.pdf

[13] Crossbow Technology mica Datasheet.

http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf

[14] Crossbow MIB520 Gateway.

http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB520_Datasheet.pdf

[15] Crossbow MIB510 Gateway.

http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB510CA_Datasheet.pd

[16] SkyeTek company homepage.

http://www.skyetek.com

[17] SkyeTek M8 RFID Reader.

http://www.skyetek.com/Portals/0/SkyeModule_M8_060524.pdf

[18] ACG RF PC Handheld Reader.

http://www.acg.de/synformation/servlet/ContentServlet/fsXzmRPA/pdfs/acg_German_Rf

[19] Linxtechnologies company homepage.

http://www.linxtechnologies.com/

[20] ANT-916-PW antenna datasheet.

http://www.antennafactor.com/documents/ANT-916-PW-QW_Data_Sheets.pdf

[21] ANT-916-JJB antenna datasheet.

http://www.antennafactor.com/documents/ANT-916-JJB-RA_Data_Sheets.pdf

[22] TinyOS Homepage.

http://www.tinyos.net

[23] Homepage of the Distributed Computing Group of the Swiss Federal Institute of Technology (ETH) in Zurich.

http://www.dcg.ethz.ch

[24] nesC 1.1 Language Reference Manual.

http://nescc.sourceforge.net/papers/nesc-ref.pdf

[25] TinyOS 1.x tutorial.

`http://www.tinyos.net/tinyos-1.x/doc/tutorial/`

[26] TinyOS programming manual.

`http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf`

[27] David Gay, Philip Levis, and David Culler. Software Design Patterns for TinyOS. To appear in ACM Transactions on Embedded Computing Systems (TECS).

`http://www.cs.berkeley.edu/~pal/pubs/patterns-tr.pdf`

[28] TinyOS Eclipse Plugin.

`http://dcg.ethz.ch/projects/tos_ide/`

[29] Mysaifu JVM.

`http://www2s.biglobe.ne.jp/~dat/java/project/jvm/index_en.html`

[30] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In ACM SIGCOMM 94 Conference on Communications Architectures, Protocols and Applications.

`http://www.cs.virginia.edu/~cl7v/cs851-papers/dsdv-sigcomm94.pdf`