NTNU
Innovation and Creativity

Practical Evaluation of IEEE 802.15.4/ ZigBee Medical Sensor Networks

Mats Skogholt Hansen
Stig Støa

Master of Science in Communication Technology
Submission date: June 2006
Supervisor: Tor Audun Ramstad, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

The wireless standard, IEEE 802.15.4/ZigBee, has been identified as a potential candidate to be used in body area sensor networks. Commercial platforms with IEEE 802.15.4/ZigBee support have been made available in the last couple of years, and an increasing number of operating systems specifically designed for network embedded systems are emerging. By implementing a sensor network application with IEEE 802.15.4/ZigBee support, experiments with actual medical data at communication rates required by medical applications shall be evaluated. By focusing on computer simulations and hardware testing of different scenarios, like in operating rooms and intensive care units, a realistic approach is to be employed. Arterial blood pressure (ABP), central veinous pressure (CVP) and electrocardiogram (ECG) data is to be used in both simulations and testing. Potential networking and physical realizations based on off-the shelf solutions are required.

Assignment given: 16. January 2006
Supervisor: Tor Audun Ramstad, IET

Preface

This Master thesis summarizes the work we have undertaken during the 10th and final semester of the Master's program at the Department of Electronics and Telecommunications, NTNU. The project was initiated by Ilangko Balasingham at The National Hospital of Norway (Rikshospitalet), and approved by NTNU with professor Tor A. Ramstad as head supervisor. Ilangko also served as teaching supervisor for the thesis. Being involved in this project has been very informative, and we have gotten an insight into the area of wireless medical sensor networks. We both enjoyed the challenging work, and would like to thank Ilangko Balasingham and Tor A. Ramstad for their support and advice.

Trondheim, June 12, 2006

Stig Støa and Mats Skogholt Hansen

Abstract

In clinical diagnostics and treatment of patients, several biological parameters have to be measured and monitored. Typical physiological parameters important to the medical staff are blood gas, invasive blood pressures, pulse rate, temperature, electrocardiogram (ECG), etc. Introducing a wireless network system for the sensor data results in greater flexibility both for the patient and for the medical staff. Wireless connections make use of digital data, which along with other digital data may enable novel clinical as well as logistics applications. With the use of computers or PDA's at other locations within the hospital, the medical staff would be able to monitor a patient regardless of his or her position as long as the he or she is connected to the network. The goal of this thesis is to investigate to what extent today's of-the-shelf sensor network platforms is suitable for biomedical sensor networks, i.e. whether the requirements of medical data gathering are fulfilled.

IEEE 802.15.4 is a wireless standard that promises great flexibility, low cost, small hardware and low power consumption. It uses the publicly available 2.4 GHz ISM band for the radio and supports a data rate of 250 kbps. Several network structures are supported, and multi-hop transmissions promise great suppleness for dynamic networks. Medical sensor data as ECG and blood pressure operate with data rates of 3.2 kbps and 1.6 kbps respectively. With these rates, a theoretical upper limit for the network size in terms of number of nodes is found to be 47 nodes. Because the radio uses the increasingly crowded ISM band, coexistence issues have been investigated. The MICAz platform is used to test examine the possibilities of the IEEE 802.15.4 standard in a medical sensor scenario. Several simulation tools are assessed, and TrueTime is found to best suit the simulation needs for the intended experiments.

The experiments carried out examine the possibility of implementing a wireless sensor network for medical sensor data. Four MICAz nodes have been set up in different experiments. Simulations are used to expand the results to a grater number of nodes. The first experiment aims to reveal the range and radio reliability or indoor and outdoor environments. The indoor experiment experiences loss of the direct line of sight component, whereas the outdoor experiment always has a strong line of sight component. Further, an experiment is set up to find the packet loss under controlled conditions with varying packet sizes and possible resends on faulty transmissions. The same experiment

is meant to uncover the effect of interference from IEEE 802.11 stations. Another experiment employs the multi-hop capabilities of the nodes. Finally a simulation is used to investigate the network breakdown in terms of network density.

The signal strength experiment revealed fluctuating results for the indoor experiment. The outdoor experiment showed that the nodes have an effective working range of 15 meters with low packet loss and about 25 meters without breaking the connection. The packet loss test for different sized packets and resends exposed a timing issue in the network access of the nodes. By using the standard defined backoff system of the CSMA/CA algorithm, large packet losses were found. No significant increase in packet loss was uncovered in the interference experiments. By implementing a preemptive backoff scheme, far better results were achieved for both the interference free and interference case. The multi-hop experiment exposed another shortcoming of the hardware. Large number of packets in the network seem to saturate the micro controller on the motes, and very irregular results were experienced. The corresponding simulation showed no sign of network saturation. The breakdown test showed that with the data rates of the medical sensors used, up to 10 nodes can be deployed before the packet error rate passes 5%. For large packets the equivalent number is 20 because of better utilization of the network resources. By implementing a simple compression algorithm known as delta encoding, the capasity of the system was doubled without any added error probability.

There are still many unsolved and unaddressed issues regarding medical sensor networks with IEEE 802.15.4. The MICAz platform is to immature for heavy network operations like routing of large amounts of data. The full potential of IEEE 802.15.4 is yet to be investigated with the beacon mode. Energy and security issues have to be addressed together with QoS. All in all, the results from the experiments point in a negative direction, but with improvements to the platform and OS together with a thought-through adaption of the standard, acceptable results may emerge.

Contents

List of Figures

List of Tables

Abbreviations

ABP	Arterial Blood Pressure
AM	TinyOS Active Message
ARQ	Automatic Repeat Request
BI	Beacon Interval
BPSK	Binary Phase Shift Keying
CAP	Contention Access Period
CCA	Clear Channel Assessment
CFP	Contention Free Period
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CVP	Central Venus Pressure
DPCM	Differential Pulse Code Modulation
DSP	Digital Signal Processor
ECG	Electrocardiogram
FFD	Full Function Device (as opposed to RFD)
GTS	Guaranteed Time Slot
IEEE	Institute of Electrical and Electronics Engineers
ISM	Industrial, Scientific and Medical
LQI	Link Quality Indicator
LR-WPAN	Low Rate Wireless Personal Area Network
MAC	Medium Access Layer
MIG	Message Interface Generator
MOTE	Collective name for wireless micro computer
O-QPSK	Offset Quadrature Phase Shift Keying
PHY	Physical Layer
QoS	Quality of Service
RFD	Reduced Function Device (as opposed to FFD)
RSSI	Received Signal Strength Indicator
WLAN	Wireless Local Area Network (IEEE 802.11 a/b/g)

Table 1: Abbreviations

Chapter 1

Introduction

In clinical diagnostics and treatment of patients, several biological parameters have to be measured and monitored. These parameters are measured by sensors attached to the patient. Typical physiological parameters important to the medical staff are blood gas, invasive blood pressures, pulse rate, temperature, electrocardiogram (ECG), etc. Another example of using biomedical sensors is for the detection of ischemia. These measurements are to a large extent done using analogue equipment, which results in a low degree of flexibility and ties the patients to the monitoring device. With the analogue solution used today, the concept of backup and information sharing is impossible. Also, with a few exceptions, the biomedical sensors are single devices with distinct outputs. This results in multiple streams of information, all in need of their own channel (i.e. cable) to facilitate measurement of multiple physiological parameters. The individual sensors are usually connected to monitoring devices via wires. This represents a potential problem; the wires tend to cause adverse events and restrict the mobility of the patient in the recovery period.

Introducing a wireless network system for the sensor data results in greater flexibility both for the patient and for the medical staff. Wireless connections make use of digital data, which along with other digital data may enable novel clinical as well as logistics applications. The data may be stored for later analysis and personnel training, or it can be utilized directly in a more effective way by the medical personnel. Using computers or PDA's at other locations within the hospital, the medical staff would be able to monitor a patient regardless of the patients position as long as the patient is connected to the network.

Advanced biomedical sensor networks may consist of several sensor nodes connected to each other in a complex manner. Using some kind of intelligent signal processing of the sensor data locally before transmitting them to a gateway would simplify and increase the efficiency of diagnostics and monitoring as well as giving the medical staff more flexibility. Processing the medical data before transmission may also result in better

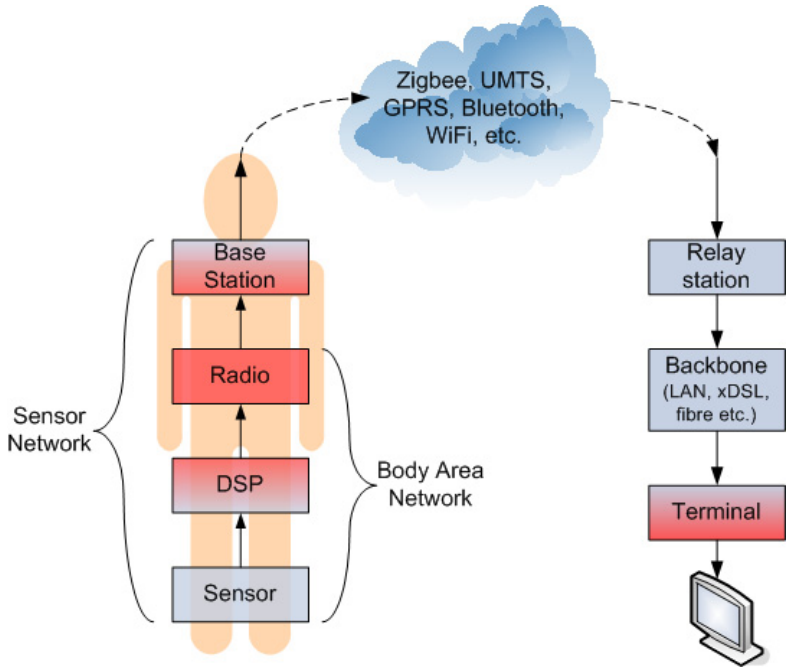


Figure 1.1: Overall system design

utilization of the available network resources. Figure 1.1 illustrates the overall system design. One patient is equipped with several sensors monitoring different parameters. The sensors may or may not have some sort of processing of the data by a *digital signal processor* (DSP) before it is sent over the air by a radio component. The sensor, the DSP and the radio component make up the *Body Area Network*. A *Sensor Network* is made up of one or more body area networks and one or more Base Stations interconnected. When the information has been gathered in the sensor network it is forwarded over the air with some wireless standard depending on the available infrastructure in the area. The information is then received at a relay station and passed on through a backbone network. In the end, the information can be viewed at terminals and monitoring stations that are connected to the network. This system has the potential of making in-home monitoring and immediate diagnostics a reality [1][2].

Being wireless, the sensor nodes are required to run on batteries. This leads to the need for a power efficient wireless network system. Some form of compression of the initial data would also be preferable to reduce the redundancy of the sensor data. This reduction would facilitate a reduced number of samples being sent to a gateway using the wireless communication protocol. This means that the life time of the batteries could be extended significantly as approximately 50% of the power from the batteries is used for data transmission [3].

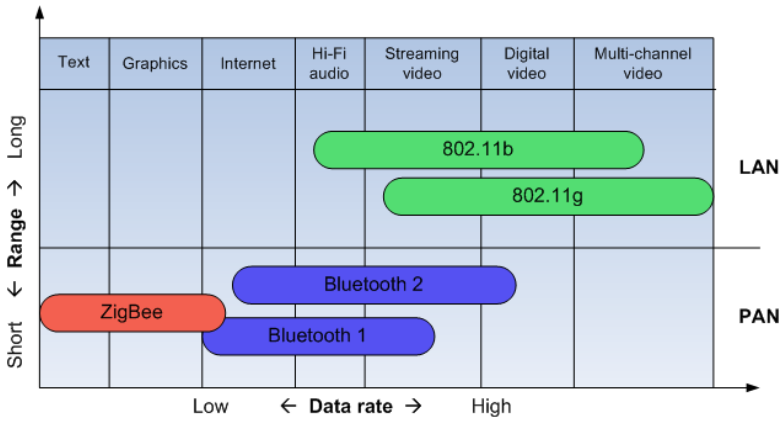


Figure 1.2: ZigBee compared to other wireless standards

A powerful Digital Signal Processor (DSP) could be implemented to do more complex analysis on the input signals. By doing so, only a keep alive signal could be used to uphold the network infrastructure. This would give an immense reduction in the data that would have to be transmitted. If a complete monitoring system for one patient included an intelligent DSP to analyze the patient status, huge improvements on the traffic load would arise. If a critical situation is detected, all patient data would be sent to monitoring stations and alarms could sound.

A wireless standard that promises great flexibility and low power consumption, for networks requiring a rather low data rate, is the IEEE 802.15.4/ZigBee standard. The standard offers low priced- and physically small hardware, and supports a vast number of devices within one network. A comparison of IEEE 802.15.4 and other wireless standards is shown in figure 1.2. The IEEE 802.15/ZigBee standard is intended for short range, but long range can be achieved with multi-hop networks and high node density. Large number of devices within a network is thus attractive for medical applications, but requires more complex network structures.

1.1 Problem

Sensor networks for medical care have a unique set of demands. They need to be extremely robust. Communication should happen with low latency, so that vital changes in the patient's medical condition can be observed immediately, for example if the system should function during an operation. The network can from time to time become very dense with a large amount of nodes having a large degree of mobility. The network must therefore be able to handle very complex radio dynamics. As opposed to a majority of today's sensor networks, medical applications demand high data rates and

several mobile observers. Sensor nodes are also expected to have the ability to recharge and energy consumption is therefore of lesser concern in medical sensor networks. As a result of these characteristics one could argue that important advances within the sensor network technology should be reevaluated. Thus, a lot of new research and development is expected before the technology could be transferred to medical applications.

The goal of this thesis is to investigate to what extent today's of-the-shelf sensor network platforms is suitable for biomedical sensor networks, i.e. whether the requirements of medical data gathering are fulfilled.

The wireless standard IEEE 802.15.4/ZigBee has been identified as a potential candidate to be used in body area sensor networks [4]. Commercial platforms with IEEE 802.15.4/ZigBee support have been made available in the last couple of years, and an increasing number of operating systems specifically designed for network embedded systems are emerging. By implementing our own sensor network application with IEEE 802.15.4/ZigBee support we will experiment with actual medical data at communication rates required by medical applications. By focusing on computer simulations and hardware testing of different scenarios, like in operating rooms and intensive care units, a realistic approach is to be employed. Actual arterial blood pressure (ABP), central veinous pressure (CVP) and electrocardiogram (ECG) data is to be used in both simulations and testing. Potential networking and physical realizations based on off-the shelf solutions are required.

1.2 Restrictions

The implementation of the complete system shown in figure 1.1 is still some distance in the future, and only the red shaded areas, or parts thereof, are of interest in this report. Below is a short discussion concerning the parts of the system that are not taken into account.

Stability and security is of utmost importance in a network system designed for medical applications. The use of wireless sensors to monitor the condition of injured patients may save lives, but if the network system fails it may result in fatalities. If the new technology is to be adopted it has to guarantee the same reliability as current systems, or better. Medical data is considered sensitive data, consequently security plays an important role in such systems. A wireless sensor network ought to implement security mechanisms that prevent unauthorized access to the medical data and mechanisms that protect against denial of service attacks. It is noted that these issues should be given great considerations in further development of wireless biosensor networks, however they are not discussed in any further detail herein.

The actual sensors that can be used to gather and digitize biometric parameters are not covered by this report. However it is worth noting that small, accurate and power efficient electronic sensor hardware is essential in the effort of creating a wireless medical

sensor network. For the experiments and simulations presented in this report, ECG, ABP and CVP sample data was supplied by The National Hospital of Norway, Rikshospitalet.

Energy consumption and energy efficiency are important issues in sensor networks. Nodes should be able to function for a prolonged amount of time without supervision and maintenance. Measuring the actual power consumption on the available test hardware is beyond the scope of this report. Yet, this issue is of great importance in the continued development of medical sensor networks. One particularly interesting area is compression. A complex compression algorithm would increase the load on the microprocessor, thus increasing the energy consumption in the processing part of the system. At the same time, reducing the amount of bits transmitted reduces the energy cost of transmission. It is very likely that there exists an ideal trade-off between the amount of energy spent on signal processing and compressing versus the energy spent on wireless transmission.

Another important restriction on the work presented in this report is the availability of equipment. For the purpose of testing, four wireless nodes have been used. This limits the ability to test the maximum performance of the system, and it limits the way inter-node interference can be investigated. All tests have been conducted with the testing nodes placed in relative favorable positions regarding line of sight and interference. The simulation part of the report is a supplement to the experimental part meant to compensate for the lack of large scale testing.

Large parts of the experiments and simulations depend on the available software. Because the IEEE 802.15.4/ZigBee standard is in ints initial deployment, testing and simulating software is immature and incomplete. Software updates for the test platform and simulation tools are continuously being developed and made available. This has resulted in numerous problems and challenges in the implementation phase, presented in chapter 3.

1.3 Structure

In chapter 2, the background theory along with a presentation of the testing equipment and simulation software is presented. The concept of compression and a look at the source data used for testing is shown in section 2.2. Later, in chapter 3 the implementations for both hardware testing and simulations are explained. Difficulties during implementation are also commented here. Chapter 4 contains the results from testing and simulation. The results are discussed and commented in chapter 5. Finally, in chapter 6, concluding remarks and prospective work is presented.

Chapter 2

Theory

This chapter deals with the background theory for the thesis. In section 2.1 the main features of the IEEE 802.15.4 standard are presented. Further, in section 2.2, medical data is presented and relevant compression theory is discussed. In section 2.3, the experimental test platform is introduced. Finally, in section 2.4, different simulation tools are evaluated.

2.1 IEEE 802.15.4/ZigBee Overview

The IEEE 802.15.4 standard is a Low-Rate Wireless Personal Area Network (LR-WPAN) standard. The ZigBee alliance was formed prior to the formation of the IEEE 802.15.4 group, but as they soon discovered, both camps were aiming at the same goal. Later, the ZigBee Alliance and the IEEE 802.15.4 group decided to join forces and ZigBee is today the commercial name for this technology [5]. However, the two groups still work on different parts of the technology. The IEEE 802.15.4 group has standardized the physical- (PHY) and the medium access control (MAC) layers [6], whereas the ZigBee alliance concentrates on the development of the upper layers and the overall development. Figure 2.1 shows the ZigBee protocol stack and the relations between IEEE 802.15.4 and the ZigBee Alliance in terms of the protocol.

The main goal of the LR-WPAN technology is quite different from other Personal Wireless Networks. Instead of offering very high rates at long distances with high QoS requirements, it is intended to serve industrial, residential and medical applications with very low power consumption and cost requirements. This can be achieved as a result of the low data rate supported; for some applications a battery lifetime of 6 months up to several years is achievable. In addition ZigBee networks may be implemented with several different flexible network structures. As shown in figure 2.5, network types as *star*, *mesh* and *cluster tree* are all supported by the standard.

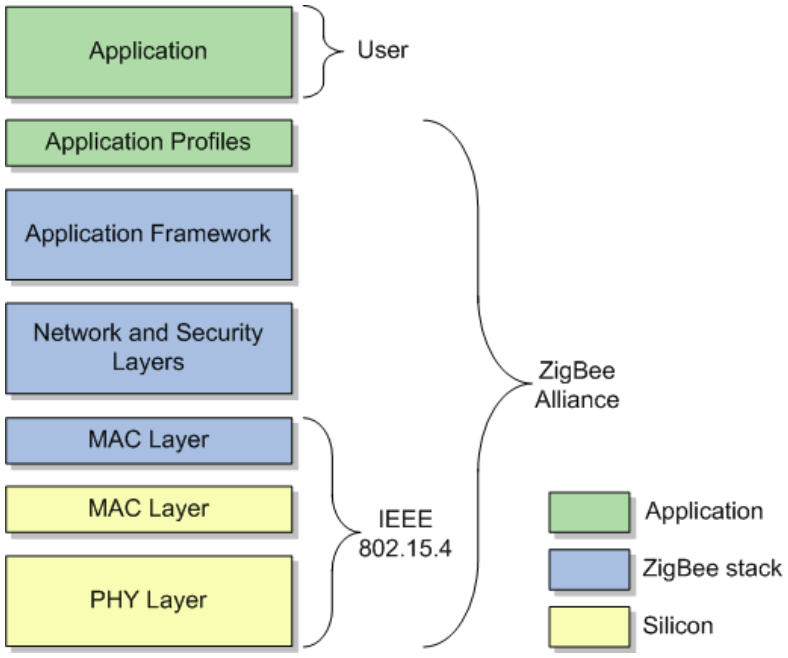


Figure 2.1: ZigBee protocol stack

2.1.1 WPAN Overview

The ZigBee network system consists of different components. The most basic is the device. A Device can either be a Full Function Device (FFD) or a Reduced Function Device (RFD). Examples are shown in figure 2.5 on page 12. A network has to have at least one FFD acting as the Personal Area Network (PAN) coordinator. A FFD can operate in three different modes: a PAN coordinator, a coordinator or a device. RFDs on the other hand are intended for very simple tasks and can only act as a device. A FFD can communicate with other FFDs or RFDs, whereas a RFD can only communicate with a FFD. Today, most available hardware is implemented as FFDs, which is also the case for the hardware used for experiments presented later herein. RFDs are intended to be even simpler, more inexpensive and more power efficient and meant for low complexity sensor applications.

2.1.2 Physical Layer (PHY)

The PHY layer consists of the data service and the management of the data service. The management part is the interface to the higher layers, and the data service part enables the transmission and reception of PHY protocol data units (PPDU) over the radio channel. The standard has two different modulation techniques, binary phase shift

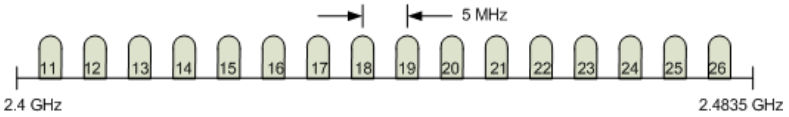


Figure 2.2: IEEE 802.15.4 operating channels in the 2.4GHz band

keying (BPSK) and offset-quadrature shift keying (O-QPSK), both used with direct sequence spread spectrum (DSSS) with a chip rate of 2 MChip/s on a 2 MHz wide frequency channel. The IEEE 802.15.4 standard operates in the unlicensed ISM bands, and the range is typically 5-75 m. Three frequency bands are supported, and have the following data rates: 250 kbps in the 2.4 GHz ISM band, 40 kbps in the 915 MHz ISM band and 20 kpbs in the 868 MHz ISM band. O-QPSK modulation is used in the first case, and BPSK in the two latter cases. The remainder of this report will only consider the 2.4 GHz band of the IEEE 802.15.4 standard because of the higher rate in this band. In the 2.4 GHz band, the spectrum is divided into 16 equally spaced frequency channels as shown in figure 2.2. Channels 1 to 11 are reserved for the lower frequency bands. The center frequency of each band can be found from

$$f_c = 2404 + 5 \cdot (k - 11) \tag{2.1}$$

where k is the channel number in the 2.4 GHz band (11-26).

The standard requires a receiver sensitivity of -85 dBm, and the defined transmit power steps are -25 dBm, -15 dBm, -10 dBm, -7 dBm, -5 dBm, -3 dBm, -1 dBm and 0 dBm. Also defined in the standard is dynamic channel selection, a scan function that steps through the supported channels in search for a beacon, receiver energy detection, link quality indication and channel switching.

2.1.3 Medium Access Layer (MAC)

The MAC layer provides service to the upper layers, and enables the transmission and reception of MAC protocol data units (MPDU) across the PHY data service. Features of the MAC layer include beacon management, channel access, *guaranteed time slots* (GTS) management, frame validation, acknowledged frame delivery, association and disassociation.

Two different modes of operation are allowed, the *beacon mode* and the *non beacon mode*. The latter is the simplest, where the coordinator does not send out a beacon. All transmissions except the acknowledgment frame have to use the unslotted CSMA-CA algorithm (see figure C.1 on page VI) to access the channel, and no GTSs are permitted. Each packet is synchronized using the preamble in the PHY packet, i.e. a device can synchronize to the coordinator by polling a data packet from the coordinator. The more sophisticated mode is the beacon mode. As illustrated in figure 2.3, the

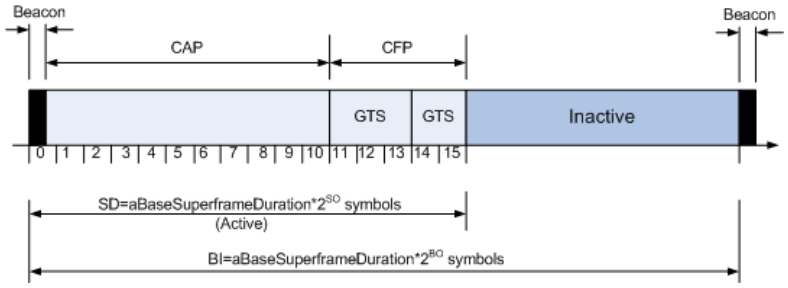


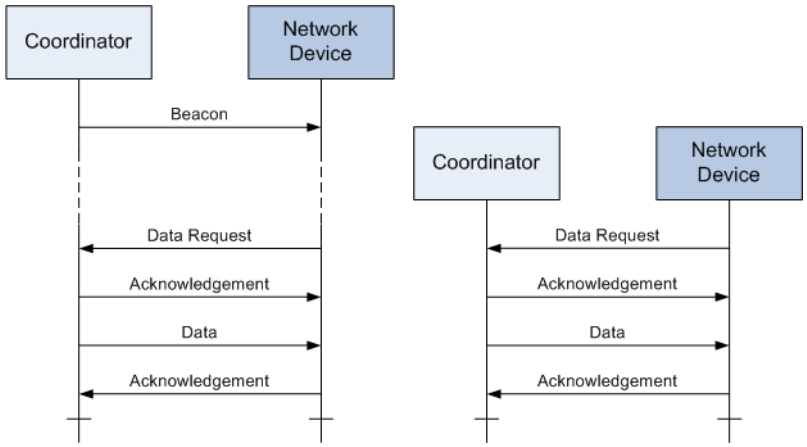
Figure 2.3: Superframe structure in beacon mode

channel is divided in time into superframes bounded by the beacon frames from the coordinator. The beacon frame is sent at the beginning of each superframe and contains synchronization information, PAN identification and information about the structure of the superframe. The superframe is further divided into an active portion and an inactive portion. In the inactive portion the coordinator does not interact with its PAN and can enter a low power mode. The active portion is again divided into a *contention access period* (CAP) and a *contention free period* (CFP). The active part is also divided into 16 equally length timeslots. In the CAP the devices waiting to transmit have to compete with the other devices using a slotted CSMA-CA mechanism (see figure C.1 on page VI). The CFP is divided into guaranteed time slots which can be allocated to the connected devices. A device may only use a GTS if it is assigned one by the coordinator, and one GTS may span more than one timeslot. A total number of seven GTS may be assigned. The CSMA-CA algorithm is more thoroughly explained in appendix C on page V.

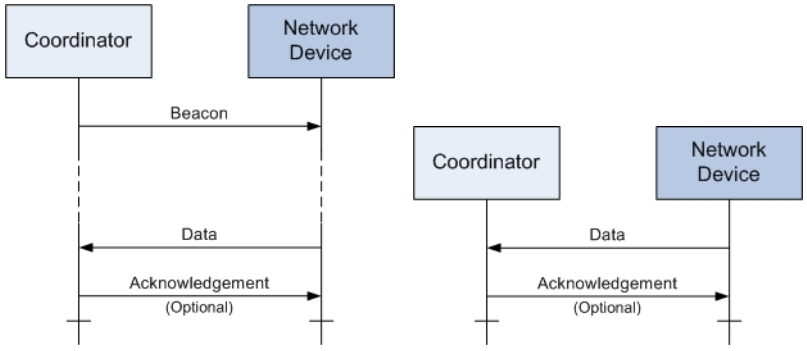
There are three main types of data transmission: from a coordinator to a device, from a device to a coordinator and from a coordinator to a coordinator. The mechanisms for these transmissions depend on the mode, i.e. beacon- or non beacon mode. In the beacon enabled mode the coordinator adds information about pending data to the devices in the beacon frame. Then the device polls the coordinator in the CAP and gets the data from the coordinator in the CAP. In the non beacon mode the devices poll the coordinator for data at an application defined rate. From devices to the coordinator the device use the slotted CSMA-CA in CAP or just send in its assigned GTS in CFP, and in non beacon mode the data is transfered using the unslotted CSMA-CA. Upon reception of data there is an optional acknowledge frame from the receiver. Four transmit modes are shown in figure 2.4.

2.1.4 Upper Layers

The upper layers of ZigBee are responsible for the routing algorithms and for the gathering of data into packets. Several different topologies are possible, and the upper layers are responsible for the routing in the network. The different network topologies have



(a) From coordinator in beacon **(b)** From coordinator in non beacon



(c) From device in beacon **(d)** From device in non beacon

Figure 2.4: Data transmission examples in IEEE 802.15.4

some distinct properties (see figure 2.5 on the following page):

- In the *Star* topology the communication runs between a single central controller, known as the PAN coordinator or Base Station, and devices. When the PAN coordinator is initialized it chooses a PAN identifier which is not used by any other network within the radio coverage.
- In *Peer to Peer* or Mesh topology there is also one PAN coordinator, but here all devices within range of one another may communicate. A Peer to Peer network may be ad hoc, self organizing and self healing. It also allows multiple hop routing of messages from any to any device in the network.
- The *Cluster Tree* topology is a special case of Peer to Peer, where most of the devices are FFDs, and RFDs are only allowed to be leave nodes. The FFDs can act as coordinators to ensure synchronization to other devices and coordinators, but there is still only one PAN coordinator.

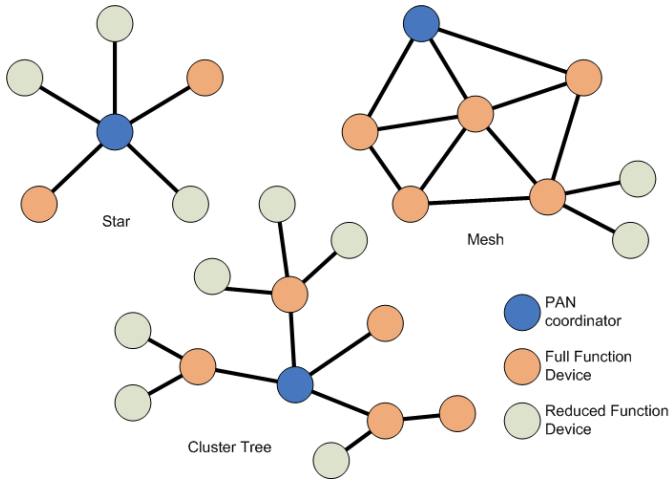


Figure 2.5: ZigBee topology models

2.1.5 Theoretical Limits

The standard states that more than 65 000 nodes are supported. However, this is just a result of the addressing space in the IEEE 802.15.4 standard. 16 bit addresses are supported which results in 65 536 possible distinct addresses. This amount of nodes in a network would put a severe restriction on the available bandwidth and transmission time for each node. In a practical system where the data rates from each node is in the order of kbit per second, that many nodes is an impossibility. One frequency channel offers 250 kbps signaling rate, but this does not take header bytes, CSMA waiting times, etc into account. The actual channel capacity, C, for for a single hop connection in a beaconless network can be found from

$$C = \frac{T_{packet}}{T_{packet} + T_{ack} + T_{header} + T_{wait}} \times C_P$$
$$T_{packet} = \frac{S_{packet}}{C_P}, T_{ack} = \frac{S_{ack}}{C_P}, T_{header} = \frac{S_{header}}{C_P} \tag{2.2}$$

where T_{packet} is the time it takes to transmit the actual data payload as shown by Tony Sun et al. in [7]. Similarly, T_{ack} and T_{header} are the respective times it takes to send the ACK packet and the headers over the air. T_{wait} is the minimum time the radio has to wait before sending a packet. Next, the equations show how to calculate the various times, where S_{packet} is the size of the payload, S_{ack} is the size of the ACK packet, S_{header} is the total size of all headers and $C_P = 250\ kbps$ is the protocol defined signaling rate.

The implementation presented in 3.2 on page 31 uses a total packet header of 30 bytes.

The ACK packets used are non addressed and have a size of 5 bytes. To get the maximum utilization of the system, i.e. to minimize the header/payload ratio, the largest possible packets of 127 bytes length is used. With a total header size of 30 bytes this gives $S_{packet} = 96\ bytes$. To find the maximum one hop capacity, the minimum wait time from CCA time, radio turnaround time and interframe spacing is used for T_{wait}. This is defined by the standard as default and reads 1.152 ms. The *upper-bound* for the effective single hop capacity for one node is then

$$\begin{aligned} C &\leq \frac{3.072\ ms}{3.072\ ms + 0.16\ ms + 0.992\ ms + 1.152\ ms} \times 250\ kbps \\ C &\leq 142.86\ kbps \end{aligned} \tag{2.3}$$

This is the maximum attainable data rate over one channel in a system with one controller and many nodes competing for channel resources, e.g. a Star network. If an information rate from each node of 3 kbps is assumed (this is an approximation for the ECG signal used later on), a maximum of 47 nodes can be accommodated for one channel. Actually, the number would in reality be lower due to the competition for resources, making the nodes wait somewhat more than they would have if they were fully synchronized and knew the sending times of all other nodes. Utilizing all 16 channels in the 2.4 GHz band, the network could theoretically facilitate 752 nodes at the most.

Now, using some sort of compression on the measured medical data would result in lower rates from the nodes, and thus increasing the number of nodes supported by the system. In a medical monitoring system, not all nodes would have the same data rate to report. Measuring temperature for instance, could be done by sending the temperature as a 5 bit digit once every 5 seconds resulting in an effective rate of 1 bit/sec, occupying only a insignificant fraction of the network resources. Also, implementing more complex network topologies with several sinks and smart channel selection algorithms would improve capacity of the system.

2.1.6 Coexistence in the 2.4 GHz ISM Band

The IEEE 802.15.4 standard makes use of the license free ISM band at 2.4 GHz. This band is free for any device to use, and one of the most widespread standards for wireless networks, IEEE 802.11 WiFi, resides in this frequency band. Another widely used wireless technology, the IEEE 802.15.1 standard known as Bluetooth, also lies in this band. Additionally, other non-networking systems, e.g. microwave ovens, may emit electromagnetic waves in the 2.4 GHz band. One of the major problems with parallel activity of different systems in one frequency band is the use of different modulation and channel access schemes. Axel Sikora et. al have in [8] studied the impact of the three most important interfering systems on the IEEE 802.15.4, i.e. IEEE 802.11, Bluetooth and microwave ovens, using real world tests of available equipment. They found that

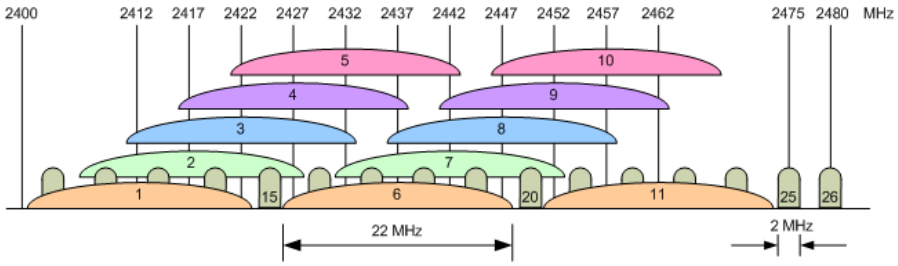


Figure 2.6: Spectrum relationship between IEEE 802.15.4 and IEEE 802.11

the impact of IEEE 802.11 stations with high traffic rate against IEEE 802.15.4 stations may be extremely critical if the same carrier frequencies are selected. Their results also show that the impact of Bluetooth is much less due to its frequency hopping scheme in the band. Nevertheless, the packet error rate increased in the order of 10%. The impact of a microwave oven was negligible when the distance from the oven was above 1 m.

A closer look at the interference from IEEE 802.11 stations can give insight into strategies to avoid some of the interference. Each frequency channel in the 802.11b standard spans for 22 MHz, and there are 11 such channels from which 3 channels are non-overlapping. So the signal of the IEEE 802.11b interferer can be modeled as band limited AWGN to the IEEE 802.15.4 signal [9]. This is due to the different bandwidths used by the two standards. As described in 2.1.2, the IEEE 802.15.4 standard employs frequency channels of 2 MHz bandwidth which is one eleventh of the IEEE 802.11b stations. Hence, the WLAN interferer would cover the entire bandwidth of the IEEE 802.15.4 device, whereas in the opposite case the interferer only affects parts of the bandwidth. Figure 2.6 shows an illustration of the frequency spectrum relationship of IEEE 802.15.4 and IEEE 802.11. Successful coexistence is best achieved if a WLAN is planned to use the non-overlapping channels 1, 6 and 11. This gives 4 channels with minimal interference to the IEEE 802.15.4 network.

2.2 Source Data And Compression

Most biological parameters are periodic over time and constrained to some interval in amplitude. A simple sensor measuring the relevant parameter will only sample the analogue input signal at a frequency, F_s, and map it to a number of predefined levels, L_k. By calculating the entropy of the now digital signal, a theoretical lower bound for the number of bits needed to represent each sample of the signal can be found [10]. The entropy of the signal can be calculated as

$$H(I) = \sum_{k=0}^{K-1} p_k log_2(\frac{1}{p_k}) \tag{2.4}$$

where $k = 0, 1, ..., K-1$ is the level number and $p_k = P(L = l_k)$ is the probability.

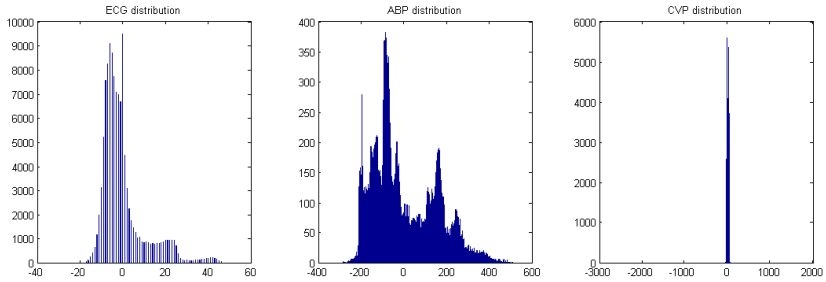
The medical data used in the experiments carried out here are from sensors used at The National Hospital of Norway [11]. ECG, ABP and CVP signals have been measured by these sensors under an experimental operation of a pig. The signals have been digitized and stored for later use. The properties of the different signals are shown in table 2.1. By doing a simple analysis on the actual data from the sensors, the distribution of the different signals can be found (see figure 2.7a). Here, the average value has been subtracted to center the values around zero.

Data type	F_s	**bit/sample**	**data rate**
ECG	200 Hz	16 bit	3.2 kbps
ABP	100 Hz	16 bit	1.6 kbps
CVP	100 Hz	16 bit	1.6 kbps

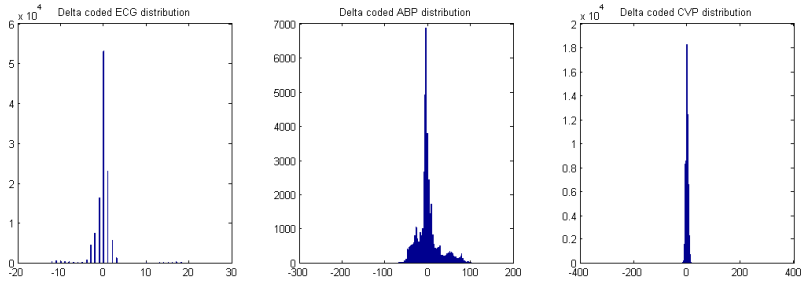
Table 2.1: Medical data properties

The entropy of the different signals was calculated using equation 2.4 and found to be $H(ECG) = 4.9\ bit/sample$, $H(ABP) = 8.9\ bit/sample$ and $H(CVP) = 5.2\ bit/sample$. This suggest that the signals are in fact quite redundant and that compression would reduce the number of bits needed per sample.

A very simple encoding strategy, known as delta encoding [12], is to send the calculated difference between two consecutive samples rather than sending the sample itself. If the change in the difference on average is smaller than the actual values, this simple strategy would yield a reduction in redundancy, and thus reduce the bit rate. The distribution of the delta encoded data is shown in figure 2.7b. By using this fairly straightforward method, the entropy of the signals are reduced to $H(ECG) = 2.7\ bit/sample$, $H(ABP) = 6.3\ bit/sample$ and $H(CVP) = 4.2\ bit/sample$ respectively. In our case, a reduction from 16 bit/sample to 8 bit/sample would give an improvement of 8 bit per sample which bisects the data rate from the sensors. The short discussion above suggests that such an improvement is within reach by using a very simple coding strategy.



(a) Distribution of input data



(b) Distribution of delta encoded data

Figure 2.7: Distribution of data

However, using the delta encoding may impose an irretrievable error. If no correction mechanisms exist, the error of loosing only one sample will propagate and increase with every lost sample. Implementing error control mechanisms is possible to reduce the effect of lost samples or packets, at the expense of lower utilization of the coding scheme. One such error control mechanism is to send a sample that is not encoded, i.e. the actual sample itself, at regular intervals. In fact, this is the encoding strategy that has been implemented (see section 3.2.8 on page 37).

More sophisticated methods for compression exist, but more sophistication is accompanied by more complexity. By using variable length codes, it is in theory possible to get as close to the entropy as needed. To expand the already suggested coding strategy, it would be possible to implement Huffman Coding [13]. Huffman coding is an entropy encoding algorithm used for lossless data compression. The term refers to the use of a variable-length code table for encoding a source symbol where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol. Implementing variable length coding also requires more error correcting mechanisms to prevent propagation of errors, hence adding overhead and complexity. As the compressing process gets more complex, it will also consume more energy. This would have to be weighted against the saved transmission energy.

2.3 MICAz - IEEE802.15.4 Research Platform

MICAz [14] is the latest contribution to the Mica family evolution [15]. Mica, released in 2001, was carefully designed to serve as a general platform for wireless sensor network research. Mica2, the successor to the Mica platform, was released one year later and corrected several of Mica's shortcomings. In 2004 MICAz was released and replaced the Chipcon CC1000 radio with the CC2420, an IEEE 802.15.4 compatible radio.

MICAz uses the Chipcon CC2420 radio in the 2,4 MHz band, a wideband radio with O-QPSK modulation with DSSS at 250kbs (see 2.1.2). The radio's higher data rates allows for shorter active periods and thereby reducing energy consumption. The CC2420 provides a number of hardware accelerators to achieve better performance. These include encryption and authentication, packet handling support, auto acknowledgments, and address decoding.

Figure 2.8: MICAz mote from Crossbow

2.3.1 TinyOS

TinyOS [16] is an operating system specifically designed for network embedded systems by the University of California at Berkeley. TinyOS has a programming model tailored for event-driven applications as well as a very small footprint (the core requires 400 bytes of code and memory combined) [17]. Among several newly developed embedded operating systems like Contiki [18], SOS [19] and Mantis [20], the TinyOS from Berkeley has recently become a de facto choice of OS on an individual sensor node, encouraging middleware components to be developed on top of TinyOS. The MICAz mote runs TinyOS 1.1.7 and higher, allowing researchers to work on the system at any level, i.e. the link protocols up to the application semantics. TinyOS' component library includes network protocols, distributed services, sensor drivers, and data acquisition tools - all of which can be used as-is or be further refined for a custom application.

2.3.2 nesC - Network Embedded System C

nesC is a systems programming language for embedded systems such as *motes*, and has been used to implement TinyOS, as well as several significant sensor applications. A mote in this context can be defined as a small wireless processing device. nesC is a component-based C dialect. Similar to objects, components encapsulate state and couple state with functionality. The principal distinction lies in their naming scope. Unlike C++ and Java objects, which refer to functions in a global namespace, nesC has a purly *local namespace*. This means that in addition to declaring the functions it implements, the component must also declare the functions it calls. A purly local namespace does not require nesC to reply to dynamic composition as required by C++ and Java. In practice, nesC often inlines a call that crosses five or six components into a flat instruction stream with no function calls. TinyOS and nesC can take this approach because sensor networks are composed of embedded computers with well-defined and tightly specified uses. By performing whole-program optimisations and compile time data race detection, nesC simplifies application development, reduces code size and eliminates several sources of potential bugs. For further reading, nesC reference manual can be found in the TinyOS document directory along with an easy tutorial. Unlike the tutorial, which is a brief introduction to get you started, *Programming TinyOS* [21] digs into nesC and how to build TinyOS applications.

2.3.3 Signal Quality Measurements

For communications IEEE 802.15.4 radios provide applications with information about the incoming signal. The effect of distance on received signal strength can be measured by the packet success rate, received signal strength indicator (RSSI) and the link quality indicator (LQI) provided by the radio. LQI is a metric introduced in IEEE 802.15.4 that measures the error in the incoming modulation of successfully received packets (packets that pass the CRC criterion).

Radio Transmission Power

Radio transmission power is programable from 0 dBm (1 mW) to -25 dBm. Lower transmission power can be advantageous to reduce interference and lower radio power consumption from 17.5 mA at full power to 8.5 mA at minimal power. RF transmission power is controlled using the TOS *CC2420Radio* library where the input parameter is an 8-bit code selected from table 2.2.

Power Register (code)	MICAz TX RF Power(dBm)
31	0
27	-1
23	-3
19	-5
15	-7
11	-10
7	-15
3	-25

Table 2.2: Transmission power control using TOS *CC*2420*Radio* library

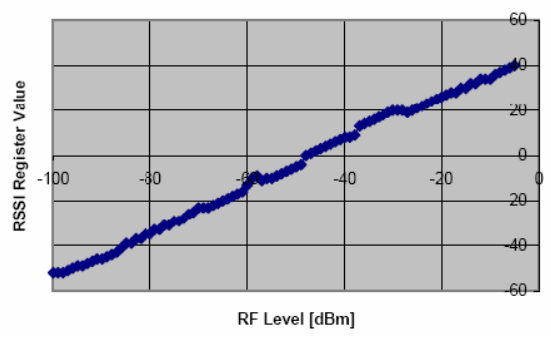


Figure 2.9: Typical RSSI value versus input RF level in dBm.

Received Signal Strength Indicator

The RF received signal strength indicator (RSSI) is read directly from the CC2420 radio. TinyOS enables easy access to the RSSI with every packet received as shown in section 3.2.4 on page 32. The RSSI value is defined by the IEEE 802.15.4 as the average value of 8 symbol periods. It is calculated based on the gain in the receiver chain in conjunction with the energy of the signal after channel filtering. Typical RSSI values for a given RF input level are shown in 2.9. The RSSI-value refers to a logarithmic scale.

Link Quality Indicator

In a wireless multi-hop network it is preferable to transmit data over links that have a high probability of success. Then the total number of retransmission required to successfully delivery data to the destination can be reduced. The IEEE 802.15.4 standard enables the measurement of link quality between neighboring nodes in the network. This measurement is in the form of a Link Quality Indicator (LQI) value that is reported with each received packet. By averaging over several LQI values an estimate of the link quality can be obtained, and therefore an estimate of the probability of successful transmission

is available to the route selection algorithm. LQI is a measure of the error in the signal, not the strength of the signal. A "weak" signal may still be a very crisp signal with no errors and thus a potentially good routing neighbor. If there is no interference from other 2.4 GHz devices, then LQI will generally be good over distance. Note that scaling the link quality to a LQI, compliant with IEEE 802.15.4, must be done by software. This can be done on basis of the RSSI value, the correlation value or a combination of those two.

The CC2420 radio datasheet [22] states that CC2420 never does chip decision. It uses a soft decision for each chip, hard decision is just done at the symbol level. The symbol decision is made based on the correlation value for each of the 16 possible symbols. The correlation value is an average over the 8 first symbols following a positive clear channel assessment (CCA), i.e. the symbol with the highest correlation value.

Bandwidth limitations in the transmitter and receiver chains will limit the correlation values to a maximum of 110 even for a perfect communication link. This is a result of the soft decision at the chip level. It is therefore not possible to directly link the correlation value to a "chip error rate". However, the correlation value will give a very good indication of the link quality, relatively independent of the RSSI level. It will e.g. give a highly reduced value in presents of a strong multipath fading, where the RSSI level still indicates a high quality link. Also, using the RSSI value directly to calculate the LQI value has the disadvantage that a narrowband interferer inside the channel bandwidth will increase the LQI value although it actually reduces the true link quality.

2.3.4 Active Messages

Radio communication in TinyOS follows the Active Message (AM) model, in which each packet on the network specifies a handler ID that will be invoked on recipient nodes. One can think of the handler ID as an integer or "port number" that is carried in the header of the message. When a message is received, the receive event associated with that handler ID is signaled. Different motes can associate different receive events with the same handler ID. The base station may send a message signaling that the motes are to reset all their states. In this case all the components would need to implement a message handler for this message type. The message handler also enables the motes to determine the message structure. For instance, the base station should be capable of receiving several different message structures. When the base station measures the RSSI and LQI values on reception, it would need to add these values to the message payload to enable the host or other motes (e.g for route estimation) to obtain them. A parameter may be expected in different positions in each message structure, and therefore the base station needs to know the current AM type to determine where to write. There is currently no management of AM. One can freely choose a number between 0 and 255, with the assumption that the AM is uniqe within the scope of its application. The active messages will be further discussed in section 3.2.4.

2.3.5 MICAz TinyOS Message Packet Structure

In TinyOS 1.x, a message buffer is a TOS_Msg (see appendix A). A buffer contains an active message (AM) packet as well as packet metadata, such as timestamps, acknowledgement bits, and signal strength if the packet was received. TOS_Msg is a fixed size structure which size is defined by the maximum AM payload length (the default is 29 bytes). Fixed sized buffers allows TinyOS 1.x to have zero-copy semantics: when a component receives a buffer, rather than copy out the contents, it can return a pointer to a new buffer that the underlying layer can use for the next received packet.

One issue that arises is what defines the TOS_Msg structure, as different link layers may require different layouts. For example, IEEE 802.15.4 radio hardware, such as the CC2420 used in the micaZ platforms, require IEEE 802.15.4 headers, while a software stack built on top of byte radios such as the CC1000 used in the mica2 platform, can specify its own packet format. This means that TOS_Msg may be different on different platforms.

The solution to this problem in TinyOS 1.x is to introduce a standard definition for TOS_Msg, which a platform can redefine to match its radio. For example, a mica2 mote uses the standard definition, which is defined in appendix A1, while on the micaZ mote a different definition is used. TOS_Msg for micaZ is defined in appendix A2.

There are two basic problems with this approach. First, exposing all of the link layer fields gives the components directly access the packet structure. This introduces dependencies between higher level components and the structure layout. For example, many network services built on top of data link layers need to know whether sent packets are acknowledged or not. They therefore check the ack field of TOS_Msg. If a link layer does not provide acknowledgements, it must still include the ack field and always set it to 0, wasting a byte of RAM per buffer.

The data payload is especially problematic. Many components refer to this field, so it has to be at a fixed offset. Depending on the underlying link layer, the header fields preceding it might have different lengths, and packet-level radios often require packets to be contiguous memory regions. Overall, these complexities make specifying the format of TOS_Msg very intricate.

Given that IEEE 802.15.4 radio interfaces are packet based, we loose considerable flexibility in software for controlling the radios operation. Since hardware accelerators are embedded in the radio instead of the micro controller, the accelerators may not be used as general purpose functions. Hence the encryption cannot be used to encrypt data buffers to be stored in flash because it is not transmitted over the radio. Also the auto acknowledgment support will prevent services to overhear messages useful for link estimation and routing because packets not addressed to the local node are discarded by hardware.

2.3.6 Micro-Controller and Flash Data Logger

The MizaZ models use Atmel's powerful Atmega128L 7.37 MHz micro-controller. The 8 bit MCU offers 128 kB program memory and 4 kB SRAM. In addition, the motes feature a 512 kB serial flash (Atmel AT45DB041) for storing data, measurements, and other user-defined information. The chip is supported in TinyOS where it is used as a micro file system. This chip is also used for over-the-air reprogramming services available in TinyOS. While new applications are installed in the mote's program memory, the 512 kB serial flash remains persistent and could therefore hold large amounts of sample data accessible to different applications. In fact, the test application described in section 3.2.1, uses the serial flash as a circular logger, emulating a medical sensor by reading data directly from the chip.

2.4 Simulation Tools

Simulating the real world opens for an inexpensive way to test a system on a large scale. In our case, only four actual motes are available for testing. To see how the system might work on a larger scale, a simulation is called for. A large number of software packages exist to simulate telecommunication systems, both wired and wireless. But, because the IEEE 802.15.4 standard still is in its initial deployment and somewhat immature, only a few software providers have made it a standard part of their simulation software. Where the standard is available, it is mostly in the development stage and/or an externally contributed part of the software. The first part of this section looks at different simulation tools that have support or limited support for the IEEE 802.15.4 standard. Next, the simulation tool used for IEEE 802.15.4 simulations in this thesis, TrueTime 1.3 [23], is introduced.

2.4.1 Simulation Software Survey

OPNET Modeler OPNET Technoligies, Inc. [24] is a provider of management software for networks and applications. OPNET Modeler is an environment for network modeling and simulation, allowing design and study of communication networks, devices, protocols and applications with great flexibility and scalability. The software may be purchased, or a software license may be obtained for free through the OPNET University Program for research and development in a non commercial setting. Upon an email price request, the cost of only the software support was reported by the sales department as

> The price for one year of technical support for a single-user license of OPNET Modeler is $2,000. The price for one year of technical support for a multi-user (30 seats) license of OPNET Modeler is $4,000.

Model development in OPNET Modeler is done through graphical model creation and C++ programming. The modeling environment is very flexible, and systems of any dimensions may be modeled. The models are divided into layers, Process model, Node Model and he complete project model. This facilitates the building of layered network models, and by creating a simple Process model and Node model, a large scale Project model may be built with few adjustments. Besides placing models in a graphical user interface, the object oriented C++ is used to program the functionality of the models at the bottom level. The OPNET model in its very core also consists of C++ code. These codes are complied and executed just like a C++ program, and enables very detailed control of the model by the user (if the user is proficient in C++). When a model has been implemented, simulation parameters can be defined and monitored during simulation. OPNET Modeler also includes analysis tools for result evaluation and comparison.

The ability to simulate wireless systems requires the Wireless Module for OPNET Modeler. However, the standard library for OPNET Modeler with the Wireless Module does not include any model for the IEEE 802.15.4 standard. An externally contributed model developed by the National Institute of Standards and Technolgy (NIST) is obtainable for free for non commercial usage at NIST's website [25]. This model is also listed on OPNET's web pages as an externally contributed model, but no guarantee is given on it's functionality or support.

OPNET Modeler is a very powerful tool for network modeling and development, and as such it is also rather complex. Building custom process and network models requires a lot of experience and practice, rather than using the already built in models which is more user friendly. OPNET is not in any way responsible for, and offers no support for the contributed models, consequently the use of such models calls for a good documentation of the contributed code and models. The IEEE 802.15.4 model supplied by NIST does not come with any form of documentation, hence implementation is a rather tedious task because of a large amount of undocumented code.

In the end, the use of OPNET Modeler as a simulation tool was dismissed due to vast problems with the contributed model for IEEE 802.15.4. The implementation of the model was considered too complex for this project. However, for future work, OPNET Modeler can be recommended if a well documented model for IEEE 802.15.4 is found or developed.

OMNeT++ OMNeT++ [26] is a discrete event simulation environment. Its primary application area is the simulation of communication networks, but because of its generic and flexible architecture, it is successfully used in other areas like the simulation of complex IT systems, queueing networks or hardware architectures as well. It's models are based on a component architecture. Components are programmed in C++, then assembled into larger components and models using a high-level language (NED). Although OMNeT++ is not a network simulator itself, it is currently gaining widespread popularity as a network simulation platform in the scientific community as well as in industrial settings. In many ways, OMNeT++ is similar to OPNET, but OMNeT++ is public-source, i.e. free for academic and non-profit use, and somewhat more limited than OPNET when it comes to supported standards and protocols.

Using OMNeT++ is very similar to the use of OPNET as described above. The same layered model is used to develop a simulation model. Here a language, NED, has been developed for the modeling on a higher level than C++. However, C++ is still the foundation of the simulation environment, thus knowledge of object oriented programming and C++ is advantageous.

There is no direct support for IEEE 802.15.4 in the OMNeT++ simulation software. Models would have to be made to fit the standard. From the community's on-line forum it can be found that the implementation is a much discussed subject, and several people

are working toward a solution. Nevertheless, a fully working contribution has not been found so far.

The use of OMNeT++ was limited due to the lack of a functioning IEEE 802.15.4 model. However, the main reason for testing OMNeT++ was to see if it had the same possibilities as OPNET. OMNeT++ is easier to get hold of and download than OPNET. The latter requires that an application is sent and the process is somewhat circumstantial. Another reason for looking at OMNeT++ was the possibility to migrate code for the actual nodes directly into the simulation environment using a software plug-in. The next section discusses one such extension.

NesCT NesCT [27] is a programming language translator that uses NesC programming language as an input and produces C++ classes for OMNeT++. NesC is the language used to implement TinyOS on the MicaZ motes described in section 2.3.2. The primary aim is to provide a new simulation environment and speed up development by using the same code in the simulation as on the actual nodes. NesCT is available for download and usage for free.

There are two main reasons why NesCT was not preferred for the simulations in this project. Firstly, the work on the simulation code and the code for the hardware motes was done simultaneously. Because of the time scale for the project it would have been very inconvenient to depend on the implementation of the actual nodes before commencing the simulation. Secondly, the use of OMNeT++ proved to be quite challenging. Nonetheless, the potential for NesCT and OMNeT++ is great since it the simulation model would be as identical to the real world test as possible.

2.4.2 TrueTime 1.3 - Simulation Tool

TrueTime [23] is a Matlab/Simulink-based simulator for real-time control systems. TrueTime facilitates co-simulation of controller task execution in real-time kernels, network transmissions, and continuous plant dynamics. It is written in C++ MEX, and is an event-based simulation. External interrupts are implemented, and it is possible to write tasks as M-files or C++ functions. Both wired and wireless networks are supported, and the wireless network block supports IEEE 802.11b WiFi and IEEE 802.15.4 ZigBee on the MAC layer level. Another feature is the possibility to make the devices battery-powered and simulate the power drain.

When installed, a separate block library, in addition to the standard Simulink library, becomes available. The TrueTime block library consists of four different blocks [28] (shown in figure 2.10):

The *TrueTime Kernel* block simulates the controll mechanisms for the nodes in the network, i.e. it acts as the micro controller in the node. The *TrueTime Network* block

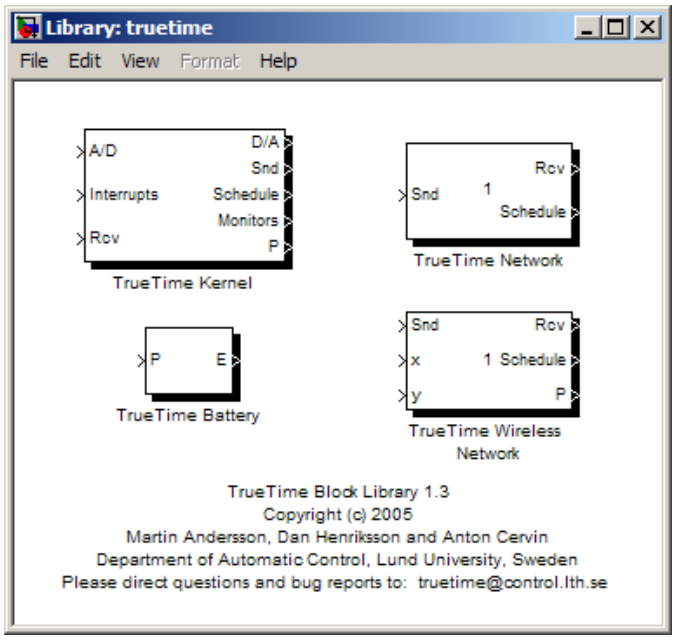


Figure 2.10: TrueTime block library

simulates medium access and packet transmissions in a local area network. Six simple models of networks are supported: CSMA/CD, CSMA/AMP, Round Robin, FDMA, TDMA and Switched Ethernet. The *TrueTime Wireless Network* block is similar to and works in the same way as the wired one. X and Y inputs are added to be able to take the path-loss of the radio into account. Two network models are supported: IEEE 802.11b/g (WLAN) and IEEE 801.15.4 (ZigBee). The *TrueTime Battery* block simulates a battery being drained and has only one parameter, the initial power. It uses simple integrator model, so it can be both charged and recharged.

Because TrueTime is based on Simulink in Matlab, knowledge of Matlab programming and Simulink simulation is beneficial. The creation of models can be done by extensive use of graphical Simulink models or by writing Matlab code for different parts of the system. The TrueTime tool comes with a user manual that describes the functions of the models and how to use the built in library. TrueTime was selected as a simulation tool because of its simplicity and, admittedly, due to the familiarity of Matlab and limited knowledge of C++ by the authors. The implementation specifications are presented in section 3.3. TrueTime is considered experimental software and can be downloaded for free from the TrueTime website [23].

Chapter 3

Implementation

Results and discussions presented in coming chapters are based on results obtained by extensive programming and testing. This chapter gives insight to the methods and equipment used by the authors. Both applications developed by the authors and tools provided by related projects have been utilized for simulations and hardware evaluation.

3.1 Experimental Setup

This chapter describes how the motes have been tested and how the system is set up for both real world testing and simulations. The equipment at hand only includes four motes, and it is therefore interesting to compare results from a real world test and a simulation. In that way the validity of the simulation model can be verified and a benchmark for the system can be found. All experiments employ the unslotted, beaconless mode. To this date, neither TinyOS nor TrueTime support the slotted beacon mode for transmission.

3.1.1 Signal Strength Experiment

Knowing the radio coverage range of the hardware is important. To test the radio range, two experiments are set up: one for an indoor environment, and one for a line of sight, outdoor environment. The received signal strength indicator (RSSI) is used as the measurement unit in the experiment. In the IEEE 802.15.4 standard, an minimum requirement is that packets received at -85 dBm RSSI shall have a maximum of 1% packet error rate. The data sheet for the CC2420 chip states that -95 dBm is a typical value for the receiver sensitivity, and that -90 dBm is a minimum value for the 1% packet error rate. For the indoor experiment the RSSI value is to be measured at different positions within 20 meters distance of a controller node. Obstacles and loss of line of sight is part of the test. The outdoor experiment is meant to establish the range of the harware

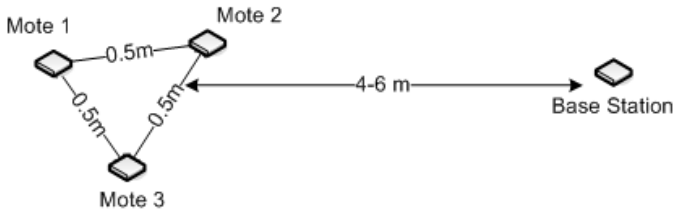


Figure 3.1: Experiment 1 physical layout

with a strong line of sight component, i.e. not an ideal open space without multipath signals. This test is not part of the simulations due to lack of propagation models in the simulation software. The simulation software is based on a simple path-loss formula with distance and a constant exponent. The purpose of the experiment is to investigate the limits of the harwdware.

3.1.2 Experiment Setup 1 and 2

In experiment 1, four motes are being used. Three motes are set up to send different medical data as shown in table 3.1, and the fourth mote is used as a collector mote that receives the signal from the three other motes and forwards it to a computer. The physical layout of the experiment is shown in figure 3.1. Other assumptions that have been made:

- All motes are static.
- Channel 26 is used (no WLAN interference).
- Non-Beacon mode, star network is deployed.
- Transmit power Tx is set to 0 dBm.

Mote	**Data type**	**(**Duration)
Mote 1	EKG	1 min
Mote 2	CVP	1 min
Mote 3	ABP	1 min

Table 3.1: Mote data in the experiments

The purpose of this test scenario is to test the performance of the system for different packet sizes and several possible retries after a packet loss. For simplicity, only a number of different packet sizes are used. More details are explained in section 3.2. The

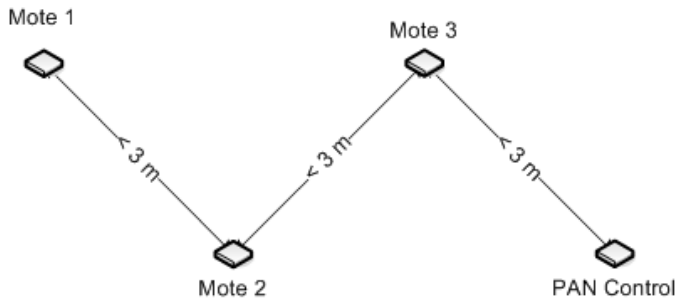


Figure 3.2: Physical layout for multi-hop experiment

measurement unit is packet loss in percent. Also a simulation is to be done for the same setup. The simulation result can be used to verify the validity of the simulation environment.

Experiment 2 has the exact same setup as experiment 1, with one variation, namely the transmit channel number. In experiment 2, a channel which has interfering WLAN stations is to be used.

3.1.3 Multi-hop Experiment Setup

The multi-hop experiment is illustrated with its physical layout in figure 3.2. It is intended to test the multi-hop properties of the system. This can be valuable for later extensions to the system with mesh networking layouts. The test is quite similar to the past tests, only now all three motes tunnel their data through a second or third mote toward the host. The simple routing algorithm is as follows. The base station has mote ID 0, and the other motes ID values 1, 2,and 3. Now each mote forwards all incoming messages to the mote with ID equal to the local ID minus one. The traffic load between mote 0 and mote 1 will thus be three times higher than with the star topology tested in experiment 1 and 2. For the experiment, the packet size is to be varied over the same range as before. The data from the three sending motes is the same as in experiment 1 (see table 3.1). All other technical assumptions for the system are also the same as in experiment 1 (see section 3.1.2 on the facing page). The experiment also gives the possibility to test forwarding functions in the motes. However, the algorithm described above does not account for routing infrastructure overhead. Here too, a simulation is set up to measure the same parameters under approximately same conditions.

3.1.4 Network Breakdown Simulation

When the number of nodes is increased the struggle for network resources increases. At some point, the traffic load eventually makes the network break down, and only a small part of the packets get through to the controller. In section 2.1.5 a theoretical limit of 47 nodes was calculated. The simulation setup for the breakdown experiment is similar to the setup for experiment 1. A star network is implemented where one controller receives all traffic, and all nodes compete for the same network resources using the CSMA/CA algorithm. The standard number of possible retries, which is 3, is used. The number of nodes range from 3 to 45 nodes in steps of three. The simulated amount of data is the same as earlier. The purpose of the simulation is to study the packet loss with increasing number of nodes for the different packet sizes.

3.2 MICAz Applications

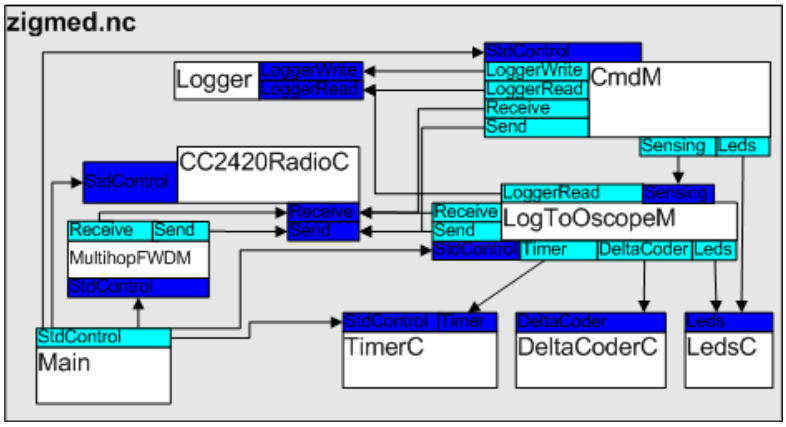


Figure 3.3: Top level configuration of the application *zigmed*

3.2.1 Software on the Motes

The application installed on the remote mote, also called the re-motes, consists of several custom components together with components from the TinyOS library. Two of the main components are the command interpreter, used to configure the motes for testing without recompiling, and the component responsible for the actual sampling and processing of sample readings. The command interpreter allows dynamic transmission power control, custom transmission offset, variable number of resends on lost packets and channel selection. Sample data is gathered from the 512 kB EEPROM to emulate sampling from medical sensors. One component performing delta coding on the sample readings and another component handling relaying of messages in a multi-hop environment, are implemented as optional modules. Figure 3.3 shows how the various components interact by the top level configuration of the application. The figure also illustrates how nesC applications wire components together, by defining both the interfaces it uses (light color) and the interfaces it provides (dark color). A more detailed system and application overview can be found in appendix E

3.2.2 Software on the Base Station

The base station consists of a MICAz mote connected to a mother board with serial connection to the host computer. The TinyOS application TOSBase is an application that acts as a simple bridge between the serial and radio links. TOSBase will copy its compiled-in group ID to messages moving from the serial link to the radio, and will filter out incoming radio messages that do not contain that group ID. TOSBase includes

queues in both directions, with a guarantee that once a message enters a queue, it will eventually exit on the other interface. The queues allow the TOSBase to handle load spikes more gracefully. TOSBase acknowledges a message arriving over the serial link only if that message was successfully enqueued for delivery to the radio link. TOSBase is modified to cope with several different channels and packet sizes. The mote is also configured to acknowledge receipt of a packet (ACK) for *automatic repeat request* (ARQ).

3.2.3 Software on the Monitoring Station (PC)

The host computer has TinyOS 1.1.15 installed. For communication with the motes, TinyOS offers an application called SerialForwarder to do the actual forwarding of packets between the base station and host computer. The SerialForwarder acts as a relay point between any packet source and network packet source. Its intended use is to provide TCP/IP connectivity to a locally connected TinyOS device. The SerialForwarder provides the capability of multiplexing multiple network connections into a single connection. That is, multiple remote applications can connect to a TinyOS node via the SerialForwarder. Tinyos also includes the *message interface generator* (MIG) for nesC. MIG is a tool to generate code that processes TinyOS messages. The tool constructs a Java object based on the original message defined by the mote application. Within the originally programmed message, the C type must be defined with $struct\ message-type$ or $union\ message-type$ in a *.h* file which is *included* by the nesC application. If the .h file defining the message type does not depend on any other files, then MIG can use this .h file directly as the message format file.

Once SerialForwarder is running. A custom Java application injects the MIG messages onto the sensor network. On the host side, another modified TinyOS application, called oscilloscope, displays the incoming messages and executes logging. The oscilloscope application is shown in figure 3.4.

3.2.4 Application Message Structure

Our application uses four Active Message types. ZigmedCmdMsg is a control message intended for the applications command interpreter. ZigmedCmdMsg and ResetOscopeMsg contains command and configuration information needed by the re-motes and is therefore only forwarded by the base station towards the re-motes. The re-motes are configured to handle the two AM types LogMsg and OscopeMsg as reply messages, in addition to receiving ZigmedCmdMsg and ResetOscopeMsg.

```
typedef struct ZigmedCmdMsg {
    int8_t seqno;               // For broadcast message comparison
    int8_t action;              // Command to schedule
    uint16_t source;            // Original source address
    uint8_t hop_count;          // Rebroadcast counter
```

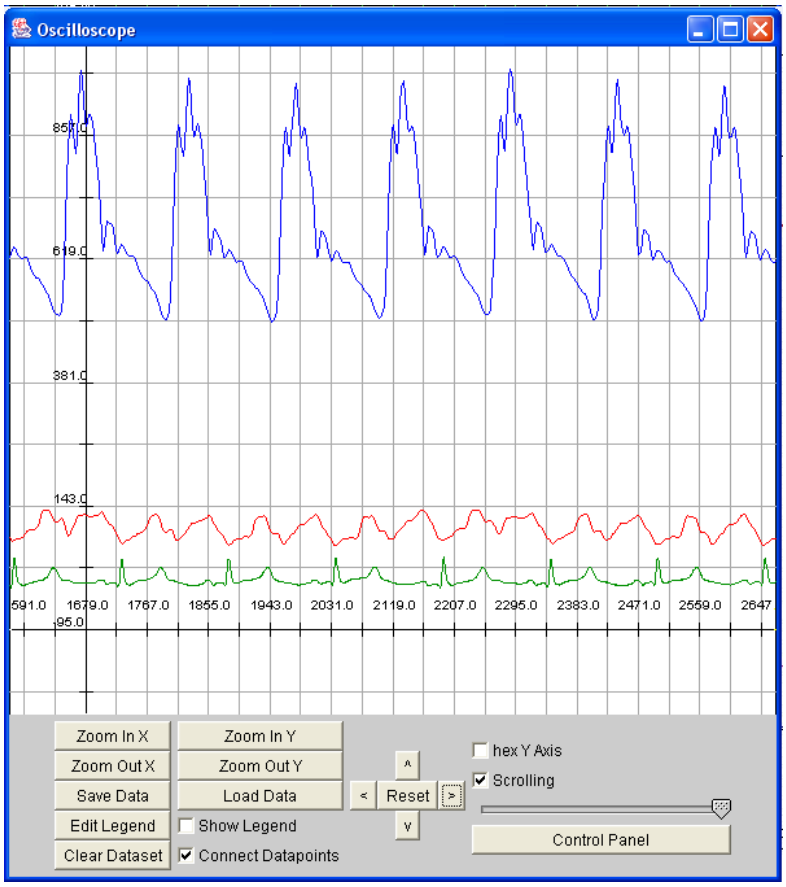


Figure 3.4: Oscilloscope application displaying sample data from tree motes

```
    union {
      start_sense_args ss_args;  // Arguments for sensor component
      read_log_args rl_args;     // Broadcast message intended for specific mote
      write_log_args wl_args;    // Values to append log
      uint8_t untyped_args[0];   // Enables variable size message in MIG
    } args;
} ZigmedCmdMsg;
```

The application command interpreter on the re-motes executes the intended command by mapping the struct member *action* to a set of predefined tasks. Since all messages from the base station are delivered by broadcast, the message also includes a sequence number to prevent redundancy and a field specifying whether the command is to be executed by all or one re-mote. The fields containing source address, hop count and sequence number are essential in multi-hop communication, both in order to build the routing tree and to control flooding.

```
typedef struct OscopeMsg {
    uint16_t sourceMoteID;      // Original source address
    uint16_t lastSampleNumber;  // Last sample number stored in message
    uint16_t channel;           // Channel given to Oscilloscope
    uint8_t rssi;               // Free space for rssi
    uint8_t lqi;                // Free space for lqi
    uint16_t data[ZBUFFERSIZE]; // Sample readings

    /* Maximum PHY packet size IEEE 802.15.4: 127                              */
    /* Maximum payload IEEE 802.15.4: 96 + (20 - address field length) = 104   */
    /* header:8(with 6 byte oscilloscope field) + buffersize*2. Min 24, Max 104 */
}OscopeMsg;
```

OscopeMsg offers the application's main communication protocol and has three components. The first three data fields enables the monitoring station to manage and display the sample data. This is an example of the cross-layer exposure of header fields discussed in section 2.3.5. The oscilloscope application on the host computer separates the motes by their *sourceMoteID* and assigns the sample data to one of ten channels. The assingnment of channels are not related to radio spectrum, rather it enables the mote to represent the signal with different rates, or to provide readings from more than one sensor. The next two data fields are intentionally left blank on the source mote. The RSSI and LQI fields are supplied or updated by the receiving mote or base station with the received signal strength and the calculated link quality indicator. This enables tracking of fluctuating radio conditions during the analysis of test results. These fields would also enable link quality aware Ad-Hoc routing. The last data field is the array containing the sample data. *ZBUFFERSIZE* is a C preprosessor defining what number of samples are to be buffered into each message. As explained in section 2.3.6, the sample data is read from the motes external EEPROM. The EEPROM always returns arrays of 16 bytes when reading from memory, thus limiting *ZBUFFERSIZE* to a multiple of eight samples. The maximum possible payload size of 127 byte, defined by IEEE 802.15.4, is attained with 8 bytes for each PHY header, MAC header and APP header together with 96 bytes for sample data. We utilize that the address header is 14 bytes less than the standards maximum recommendation, offering additional bytes to the payload.

```
typedef struct LogMsg {
    uint16_t sourceaddr;        // Original source address
    uint8_t load_size;          // Maximum payload size
    uint8_t chnl;               // Current active channel (11 - 26)
    uint8_t txpower;            // Motes current transmission power
    uint8_t state;              // Number of allowable retransmissions
    uint8_t rssi;               // Free space for rssi
    uint8_t lqi;                // Free space for lqi
    uint8_t log[16];            // The first 16 bytes from the log
                                // containing sample data type
} LogMsg;
```

LogMsg is used purely as a way to verify the re-motes current state.

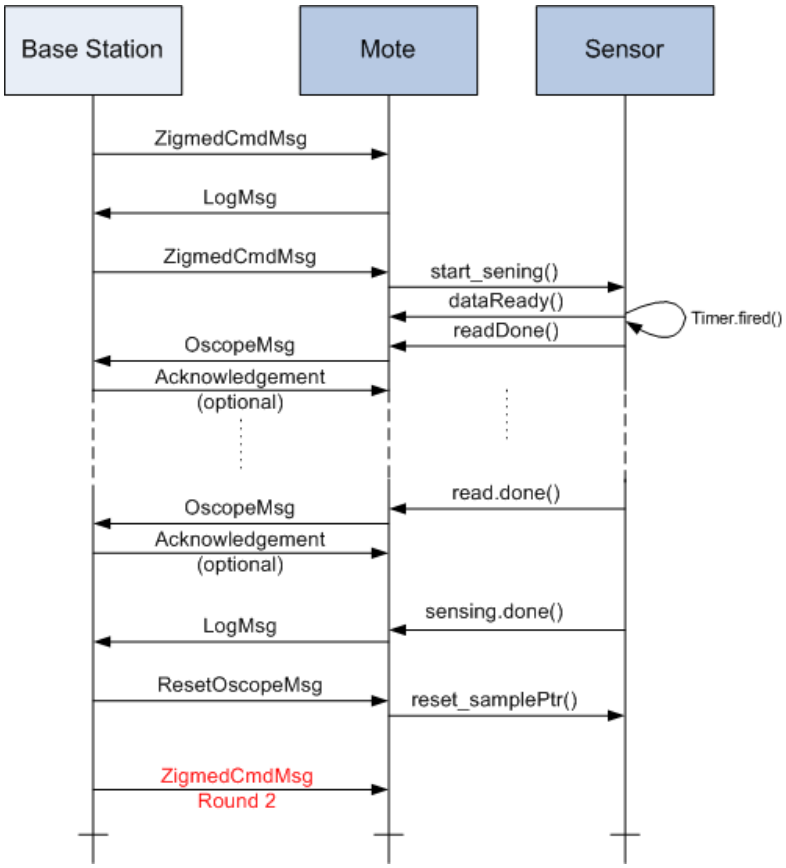


Figure 3.5: Test application float diagram

3.2.5 Test Application

A Java application was written to execute a set of predefined tests and manage the data logging. The application establishes a connection to the TOSBase by running a SerialForwarder. Once the SerialForwarder is running, the oscilloscope application starts up and begins to listen for packets with AM types OscopeMsg and LogMsg. Figure 3.5 illustrates the test applications message handling. Note that the different occurrences of ZigmedCmdMsg has different functionality (see appendix B for details). To verify correct re-mote configuration during each test run, a LogMsg is now requested from each re-mote and the resulting message contents is written to the beginning of the current log file. When all re-motes have replied, one minute of sample data is requested and appended to the data log on the monitoring host.

3.2.6 Radio Reliability

The main bottleneck on our testbed is the 57.6 kbaud serial connection to the host, which in theory could handle a maximum of 350 packets of size 41 byte per second. However, in practice it does not. Quoted from Crossbow INC's website:

> The DMU works very reliably at 38.4K baud. We have tested it to be working fine up to 57.6K baud. At 115.2K baud, sometimes it works and sometimes it does not, depending on the configuration of your PC, OS etc.

The base station running TOSBase is no more than a MICAz mote with a serial connection to the host computer. Therefore the limited computational performance and the fact that the mote can not send and receive at the same time must be accounted for. Especially since the base station needs to kick back and forth between the two during acknowledgments.

A task post failure in the serial or radio stacks may cause the TOSBase to hang. The chance of this is lessened by including the line CFLAGS += -DTOSH_MAX_TASKS_LOG2=8 line in the TOSBase Makefile, increasing the size of the task queue to 256.

3.2.7 Implementation Notes

Over the Air Programming Among several TinyOS library functionalities tested during development but not implemented in the final test application, one stands out in particular. The Deluge application provides an efficient method for disseminating large data objects, such as program binaries, to many nodes within a wireless sensor network. Combining this mechanism with a bootloader and command dissemination allows for the wireless programming of nodes. Some of the features included with Deluge are multi-hop support and epidemic propagation. Multi-hop support allows wireless programing of all nodes in a multi-hop network. The epidemic continuous propagation effort by all nodes then helps ensure reachability of those nodes with intermittent connectivity. The functionality of over the air programming and network configuration has a great potential and should have a great impact on the sensor network questioned by this thesis. The main reason for not implementing Deluge is the frequent need for flash formating upon reinstalling. The current external EEPROM, used by Deluge for bootable images, already contained sample data and would have to be re-written or at least verified after every new install.

TinySec Encrypted Messages In earlier versions of MICA, encryption was done in software by the TinyOS library *TinySec*. IEEE 802.15.4 defines 128 bit AES encryption implemented in hardware. On the MICAz mote, TinyOS enables hardware encryption by replacing the message buffer TOS_Msg with TinySec_Msg. This should be transparent

to the application except for the reduced maximum payload. Maximum message payload with encryption is then 87 bytes.

At present, the software used for programming and testing our applications are rapidly evolving, and for every new release parts of the code library are deprecated. Our application accesses the EEPROM by using the Logger component in TinyOS. This component has now become deprecated and further development should consider the use of a newer abstraction to the EEPROM hardware. However, the functionality of Logger was used to simulate medical sensors, and is therefor no longer needed once the application is developed.

3.2.8 Compression

The mote's application described in chapter 3.2.1 also implements a simple compression algorithm. The compression is in the form of a delta encoding as described earlier in chapter 2.2. To illustrate how powerful this simple compression technique is, let us first look at 16 subsequent samples from the ABP and ECG sample data:

```
ABP:
867  873  849  821  824  863  914  951  954  924  884  859  856  864  868  862
(radix 10)
0363 0369 0351 0335 0338 035F 0392 03B7 03BA 039C 0374 035B 0358 0360 0364 035E
(radix 16)

ECG:
-7   -7   -7   -7   -6   -6   -5   -4   -4   -3   -3   -2   -2   -1   -1   0
(radix 10)
FFF9 FFF9 FFF9 FFF9 FFFA FFFA FFFB FFFC FFFC FFFD FFFD FFFE FFFE FFFF FFFF 0000
(radix 16)
```

After encoding:

```
ABT:
867  6    -24  -28  03   39   51    37   3   -30  -40   -25  -3   8   4   -6
(radix 10)
0363 06   E8   E4   03   27   33    25   03  E2   D8    E7   FD   08  04  FA
(radix 16)

ECG:
-7   0    0    0    1    0    1     1    0   1    0     1    0    1   0   1
(radix 10)
FFF9 00   00   00   01   00   01    01   00  01   00    01   00   01  0   01
(radix 16, little endian)
```

From the ABP sample data we see that large integers can be represented with substantially smaller integers after encoding, and thereby reducing the sample from two bytes to one byte. The ECG sample data also show small variations in the sampled signal, keeping the differential of the signal close to zero. This is exactly what was predicted in section 2.2 (see figure 2.7).

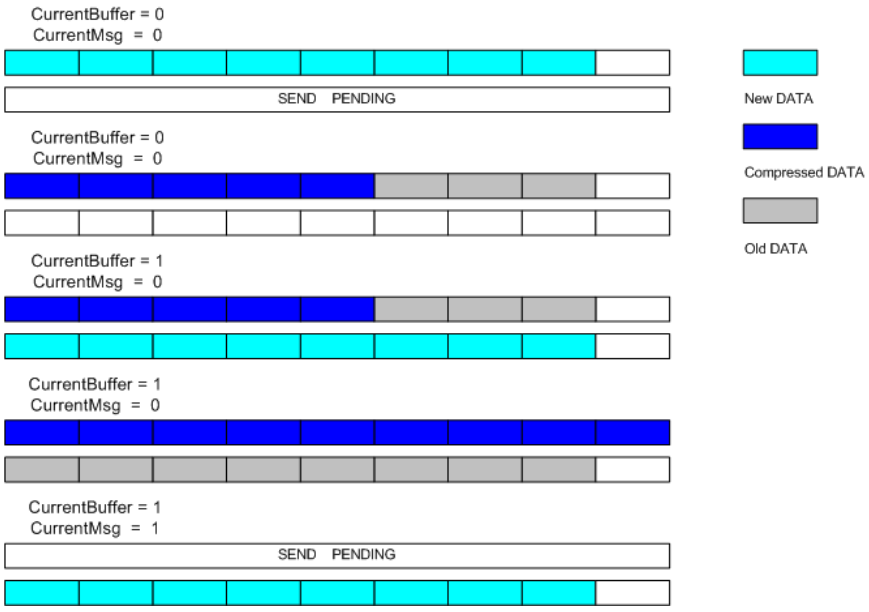


Figure 3.6: Buffer usage illustrating compression of sample data.

Mote sensors sample at predefined intervals and would therefor need buffer access whenever a new sample is ready. In order to secure messages from alteration while waiting for channel access and successful transmission, two separate message buffers are implemented. This property makes a good foundation for message compression as well. By utilizing the two buffers, it is possible to read sample data from the sensor, compress data and wait for successful transmission. This is done without adding additional buffers, changing the message structures (apart from appending one extra byte) or changing the properties of the original application. Figure 3.6 illustrates the buffer usage with the use of compression.

The additional byte introduced by the compression algorithm eliminates the propagating error problem as a consequence of delta coding. In general the algorithm will always be vulnerable to propagating errors as long as the samples are dependent on each other. The statement is nevertheless valid for the higher levels of the application where it is intended. This property is a direct consequence of the cyclic redundancy check done at the link layer. Packets that do not pass the CRC test are dropped, and all packets received for delta decoding on higher layers will therefor not contain single sample errors assuming it was encoded correctly.

3.3 Simulation with TrueTime

The simulations were implemented in a Simulink environment using the models in the TrueTime block library. All other blocks used to create the models were taken from the standard Simulink library. All node functionality was programmed in Matlab *.m* files. Although node mobility is supported in TrueTime, the nodes remained stationary during all simulations to be able to see the effects of different parameters and network sizes more clearly.

3.3.1 Simulating IEEE 802.15.4 Networks with TrueTime

The TrueTime Wireless Network block is the central part of the simulation model. It gathers the outgoing signals and positions from all nodes and calculates the respective inputs on the wireless channel for all nodes. To control the functionality of the network, a set of parameters have to be set:
Network number is a unique number for each network in the system. Different networks do not affect each other in the simulations.
Number of nodes, within the network.
Data rate is the rate of the transmitted bytes which controls the duration of a packet transmission.
Minimum frame size represents the minimum size of a packet, i.e. the total number of header bytes. Packets that are smaller than this will be padded.
Transmit power determines how far the signal will propagate (given in dBm).
Receiver signal threshold If the received energy is above this value, the medium is considered busy.
Path-loss exponent is used to calculate the loss of the signal from sender to receiver. The path loss is modeled as $\frac{1}{d^a}$ where d is the distance and a is the chosen exponent.
ACK timeout is the time a sending node will wait for an ACK message before concluding that the message was lost.
Retry limit is the maximum number of times a node shall try to retransmit a message before giving up.
Error coding threshold is a number in the interval [0,1] which defines the percentage of block errors in a message that the coding can handle.

All nodes, including the network controller, are implemented using the TrueTime Kernel block from the TrueTime library. The TrueTime Kernel is initialized with a piece of Matlab code where the node number and the number of input and output ports are set. A periodic task and an interrupt handler is also defined in the initialization script. Each kernel block also has the options of *battery* powered on or off, *clock drift* and *clock offset*. The actual connections between the nodes and the network and different inputs/outputs are set using the graphical interface of Simulink.

Parameter	Setup 1	Setup 2	Setup 3
Network number	1	1	1
Number of nodes	4	4	4...46
Data rate [bps]	250k	250k	250k
Minimum frame size [byte]	30	30	30
Transmit power [dBm]	0	0	0
Receiver signal threshold [dbm]	-85	-85	-85
Path-loss exponent	5.5	5.5	5.5
ACK timeout [sec]	8.64e-4	8.64e-4	8.64e-4
Retry limit	3	0...9	3
Error coding threshold	0	0	0
Packet size [byte]	46, 62, 78, 94, 110 and 126		

Table 3.2: Parameters for simulations

3.3.2 The Simulation Setup

Three different setups were used for the simulations. The first setup is used to experiment with TrueTime and to investigate the possibilities for sending data through the network. Actual sensor data from files is used at the sensor end, and the controller has a scope connected for monitoring purposes. Three sensor nodes and one controller make up the network. Figure 3.7 shows the setup in the graphical setup in Simulink. To measure the loss of packets, all outgoing packets are marked with a sequence number. On the receiver side, all lost packets are counted, i.e. whenever the incoming sequence numbers are out of order, a loss is registered. The positions of the nodes are displayed using a Mote Animation block. Tabel 3.2 summarizes the network parameter setup.

The second simulation setup is to a large extent the same as the one shown in figure 3.7. Only now the actual source data is replaced with dummy packets of different sizes. This does not effect the results because the actual content of the packets is of no interest when packet loss is considered (more on this in section 4.1). However, using dummy packets instead of real data reduces the computational complexity for the simulations. The purpose of this simulation setup is to find the packet loss of the system for different packet sizes and different number of possible retransmissions. This result is going to be used as frame of reference against the hardware test to examine the validity of the simulation model. The same model is used for the multi-hop system test.

Finally, the third model is revealed in figure 3.8. The model is essentially the same as the previous model, but now three and three nodes have been grouped in subsystems and a total of 15 subsystems are placed in the system, i.e. a total of 45 sensing nodes in addition to the controller. Packet loss is the measured parameter here too, only this time the breakdown of the network is investigated according to the number of nodes

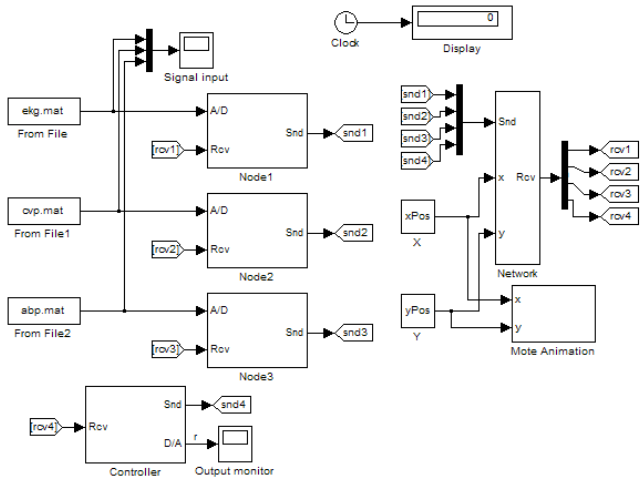


Figure 3.7: Simulation setup in Simulink for monitoring test

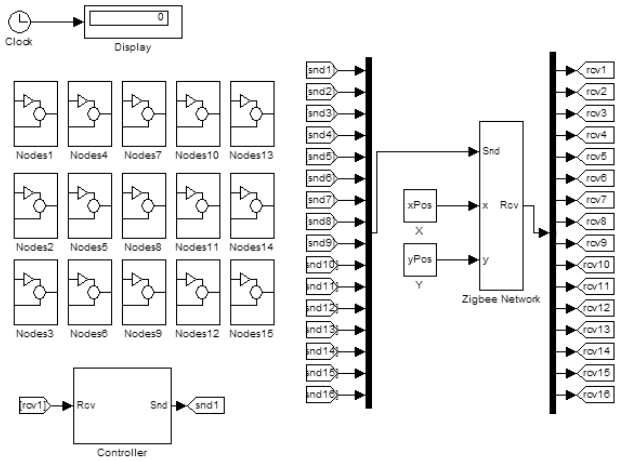
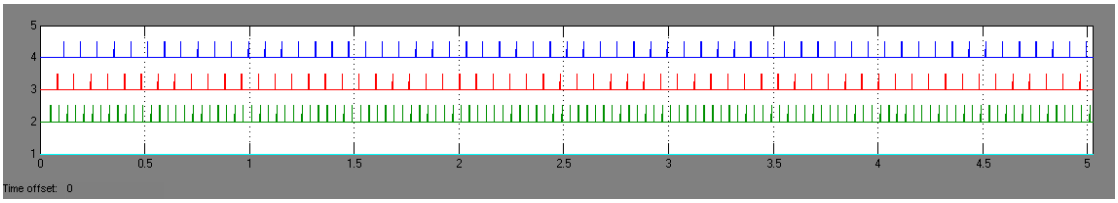


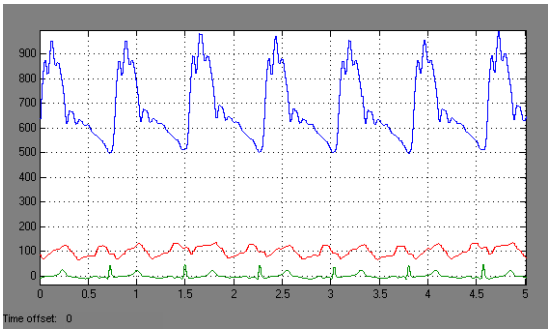
Figure 3.8: Simulation setup in Simulink for 45 nodes

in the network. The number of nodes in the network is varied by adding or removing subsystems of three nodes.

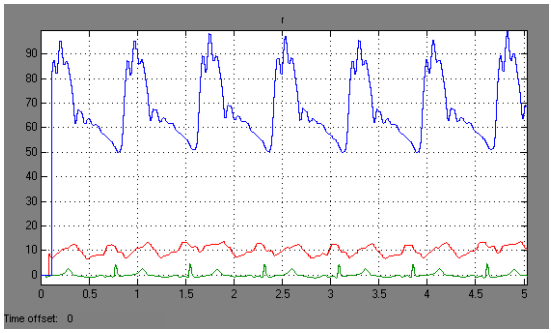
In the first simulation, real data is sampled and sent over the simulated network. Figure 3.9 shows the different views of the simulations. The network schedule illustrates the different nodes access of the channel. Signal input and signal output is shown in figure 3.9b and c respectively. The sampling delay, i.e. the time interval used to fill a packet, is clearly visible in the output view. In figure 3.10a, the simulated layout of



(a) Network schedule

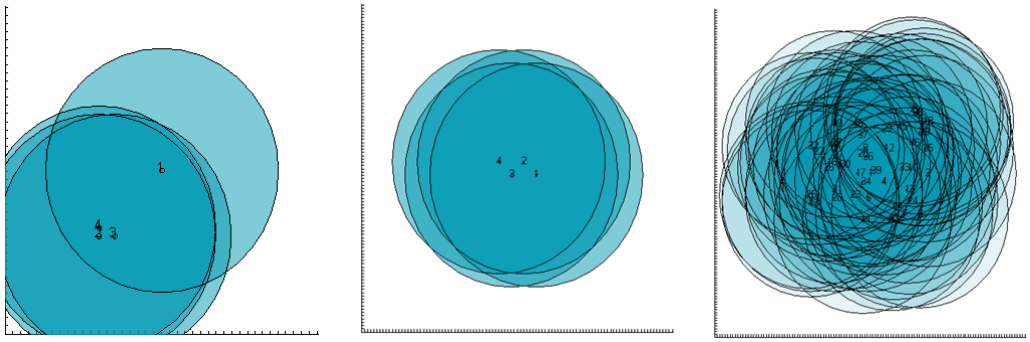


(b) Signal input



(c) Signal output

Figure 3.9: Simulation views



(a) 3 nodes static **(b)** Multihop simulation **(c)** 45 nodes random placement

Figure 3.10: Node placement and coverage for simulation

the nodes is shown. Figure 3.10 b and c show the placement for the multi-hop test and breakdown test respectively.

Chapter 4

Results

In the following chapter, results gained from testing and simulation are presented. The results from experiments and simulations were saved to files and analyzed using Matlab scripts.

4.1 Test Results

For the system implemented, a CRC criterion is used to decide whether a packet was received correctly or not. Hence, for the nodes being tested, errors on the actual data will not occur. The only source of error is when a packet for some reason is lost. However, there are three different ways for a packet to be corrupted and thereby to be considered lost at the receiver end. Firstly, a packet may be lost due to errors in the wireless transmission which results in a bad CRC or not even a reception at all. The second possibility is that two nodes send their packets at times so that the transmissions overlap in time (referred to as the hidden station problem [12]), resulting in two lost packets. Finally, a packet may be lost before it is ever sent. This is possible if a node senses a channel as busy a maximum number of times (user defined within the values 0 to 5). In this situation, the node will simply discard the packet and move on to the next. Finding the source of the packet loss is complicated in terms of the hardware at hand and not considered vital for the following results and discussion.

4.1.1 Signal Strength (RSSI)

The received signal strength was measured inside and outside the university building and the results for the indoor experiment are shown in figure 4.1. The base station, marked as a white dot to the center left of the figure, was placed at shoulder height and the re-motes 20 cm below and at the same height as the reading stalls. The measure

points are marked with X in the figure. The 2 meter high lockers in the bottom right corner of the reading room are made of metal, and block the signal completely if the mote is placed inside.

An experiment was also carried out on a parking area outside the university building and the results are shown in figure 4.2. Within a 20 meter distance from the university building on one side of the experiment and a few cars parked nearby, the effect of multipath fading was still present, i.e. with no fading a log normal curve could be expected because of a strong line of sight component. We chose to do the experiment near the university building and not in an open area, e.g. a football field, since this is a more likely real world scenario. We observed that within 25 meters of the base station the RSSI of small packets were generally stronger than with the larger packets. For longer distances the larger packets gave stronger RSSI. In fact the small packets lost connection completely at 30 meters distance, while the large packets lost connection after 40 meters. Although packets were delivered correctly at 40 meters, the connection broke sporadicly. The effective working distance was 15 meters with a small amount of packet loss and, 25 meters without braking the connection. Note that the values presented in figure 4.2 are mean values from the packets that *were received* and do not imply a steady link.

The remaining experiments were performed with stationary nodes. Still, the RSSI values have a tendency to fluctuate. Figure 4.3 shows the RSSI values during a typical 60 seconds experiment.

4.1.2 Result Experiment 1

Figure 4.4 shows the test results for experiment 1 (defined in 3.1). Each point in the figure represents the average of 5 separate experiments, each with a duration of 60 seconds. All tests have been conducted by sending data in real time. The packet loss was measured for different packet sizes and number of possible retries. A retry may be sent if the packet is not received or an ACK is lost. Channel number 26 was used to avoid interference from IEEE 802.11 WLAN devices. From equation 2.1, the center frequency at channel 26 is 2479 MHz. The highest frequency channel for IEEE 802.11b/g is located in the range 2451 MHz to 2473 MHz. The figure shows that the packet loss declines with increasing number of retry possibilities. This result was anticipated, as the packets are sent relative infrequently. With more retry possibilities, the chance of a successful transmission increases. In the other dimension, different packet sizes are shown. From the figure it is evident that packet size is of less significance to the packet loss than number of retries. However, there is a tendency that minimum packet size, i.e. 46 bytes, results in a higher packet loss than medium sized packets. Maximum size packets also have a small tendency of higher packet loss. One element that is not shown in the figure is the loss of ACK packets. If a packet was received correct, and the belonging ACK packet is lost, the original packet is resent and thus, possibly received two or more times.

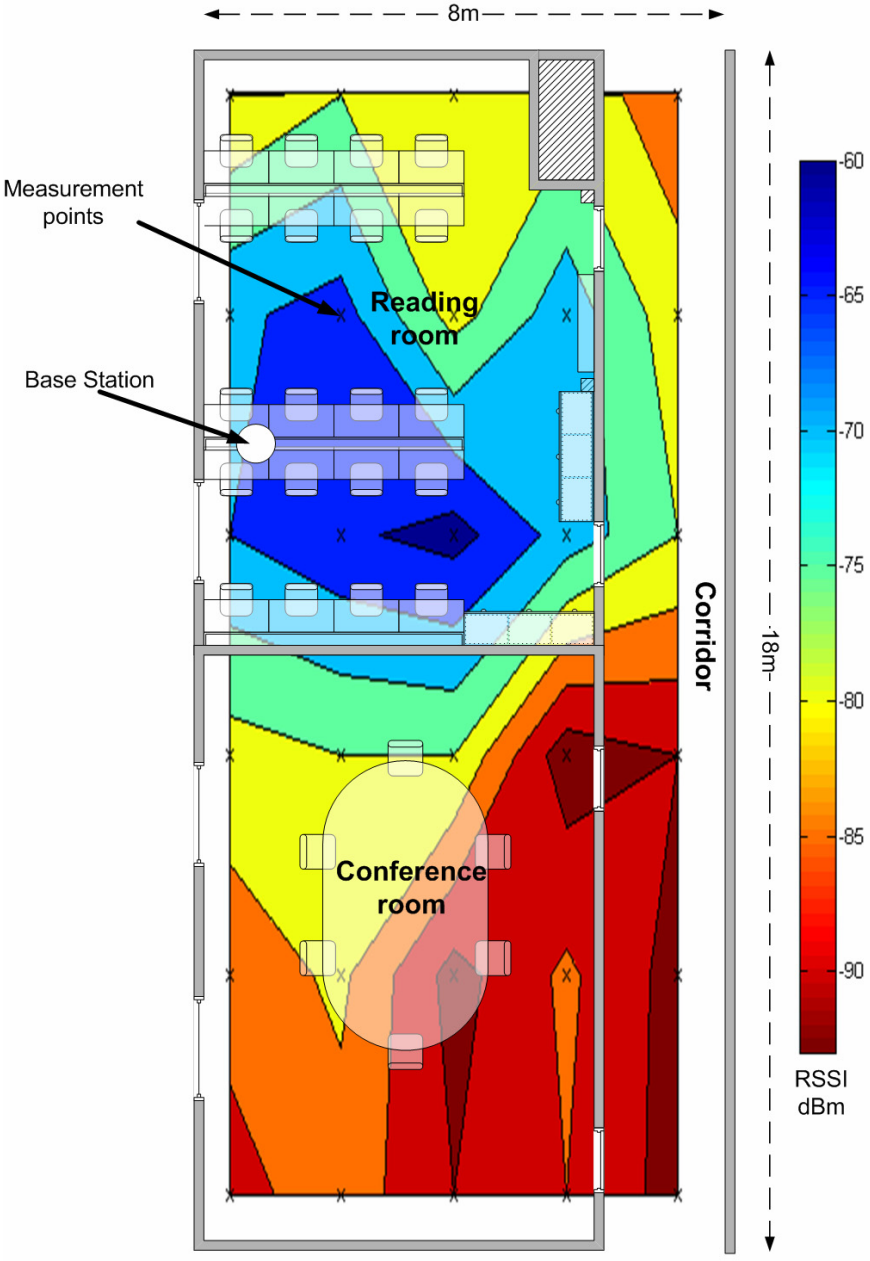


Figure 4.1: RSSI values measured indoors

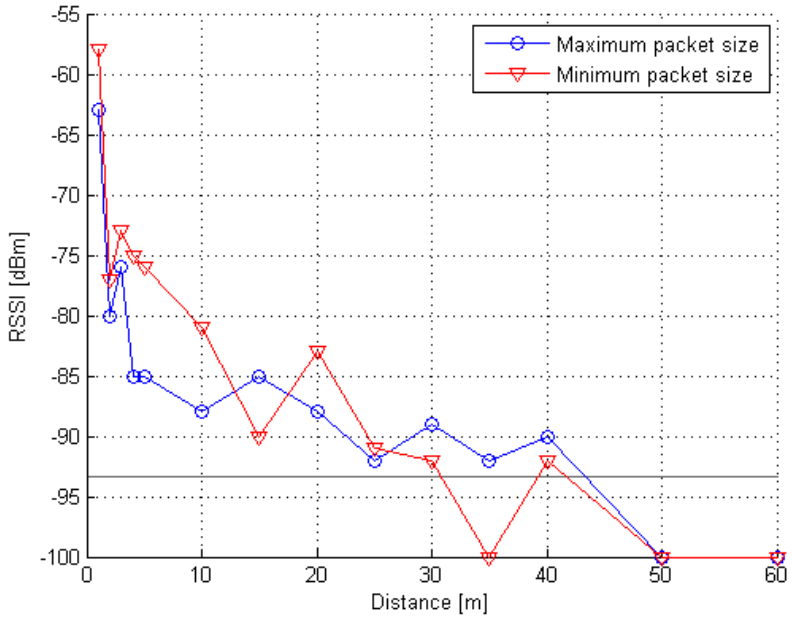


Figure 4.2: RSSI values measured outside

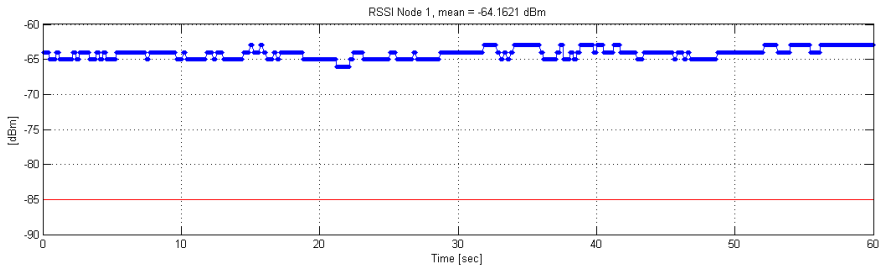


Figure 4.3: Typical RSSI values for a 60 seconds experiment

The worst result occurs at 46 bytes packet size and 2 reties and reads about 10% loss. The best case lies at a packet size of 78 bytes and the maximum number of possible retries (nine) and reads appoximately 2% loss. These values are far worse than what was expected. However, the timing issue at the beginning of the sampling may be the source of the problem.

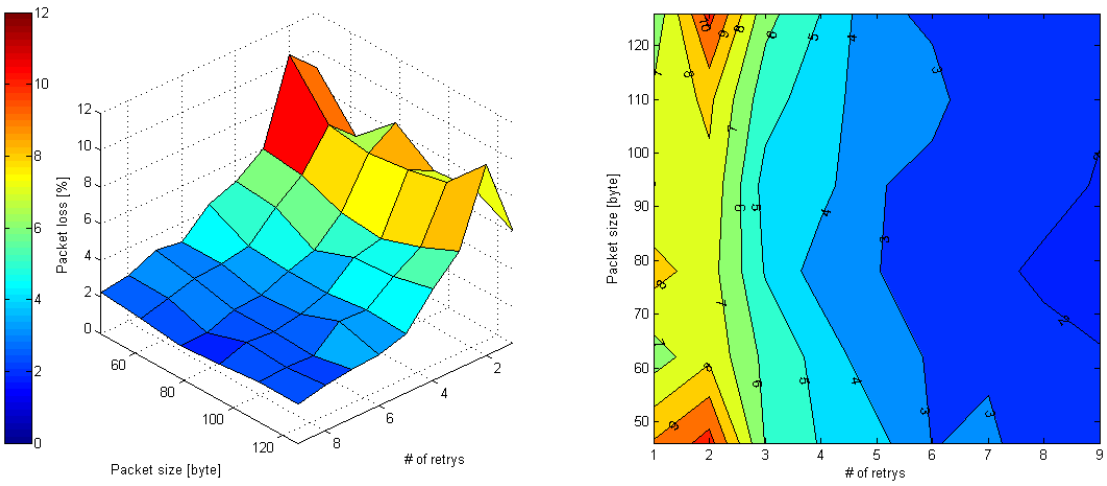


Figure 4.4: Test result no interference

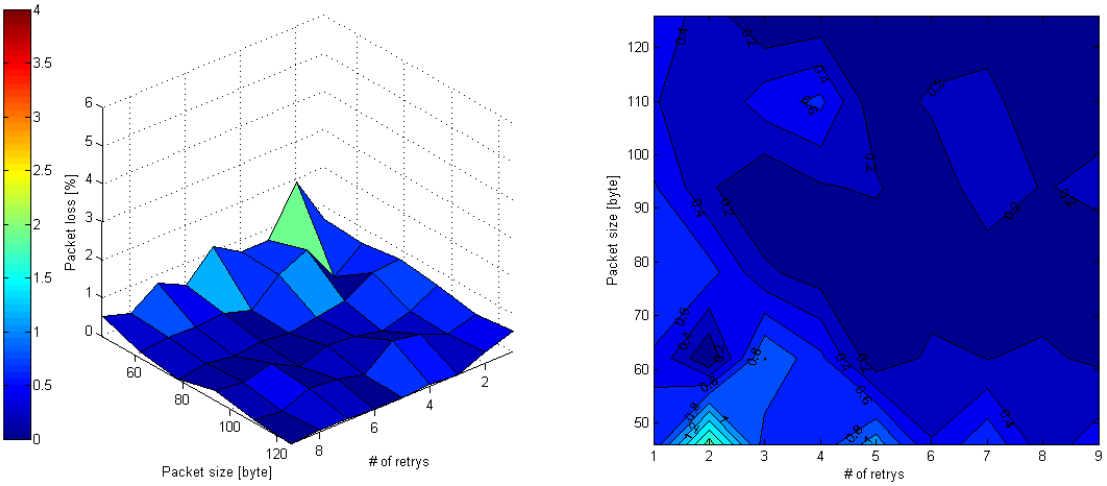


Figure 4.5: Test result no interference, using preemptive backoff

4.1.3 Node startup timing

In figure 4.6, a coarse illustration of the network timing for three nodes is illustrated, given that all three node sample the same type of signal, i.e. all three nodes spend the same amount of time gathering enough samples to fill a packet. At time t_0 the three nodes get instructions to start measuring and sending data. They start sampling data at the same time. At time t_1, all nodes will have a packet ready for transmission, and they have to compete for the channel using the CSMA-CA algorithm. This may result

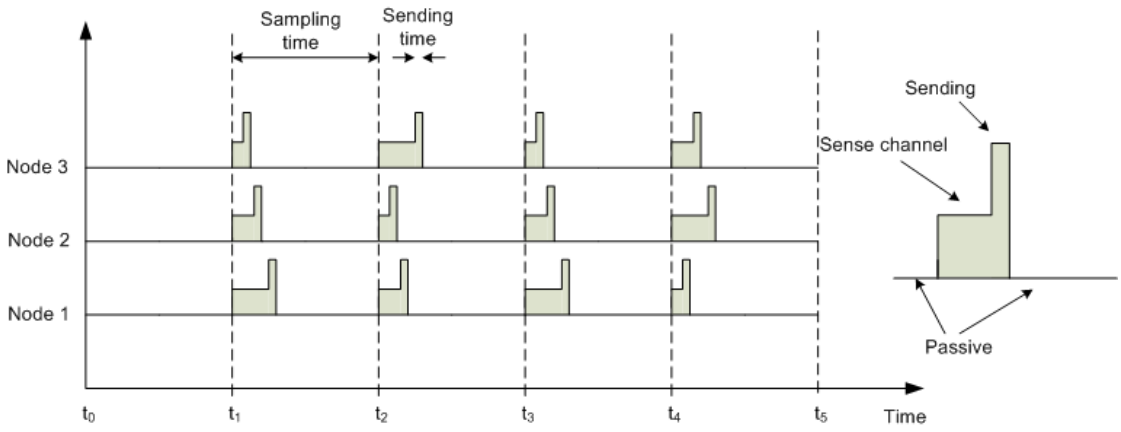


Figure 4.6: Network traffic, standard setup

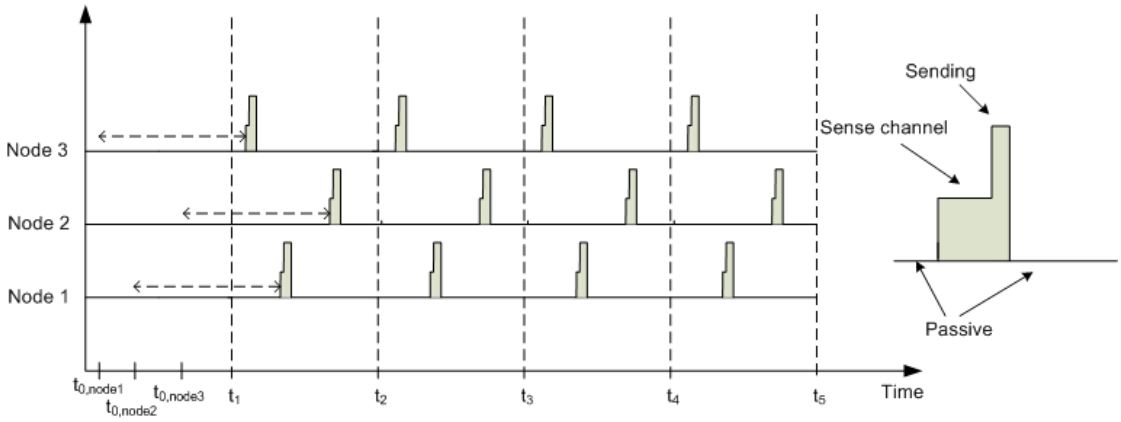


Figure 4.7: Network traffic, custom backoff setup

in collisions and dropped packets. The sampling for a new packet however, continues at the same pace as before, and regardless of who got to send a packet first, all three nodes are ready to send a packet again at time t_2. Again the three nodes have to compete for channel access.

In the ECG case for the minimal packet size, the sampling time equals the number of samples in the packet divided by the frequency of the source signal, i.e. $t_{sampling} = 8samples/200Hz = 40\ ms$. The transmission time for the packet is $t_{send} = 46bytes * 8/250000bps = 1.472\ ms$ (8 samples*2 bytes+30 byte overhead=46 byte). The sampling time is in the order of 30 times larger than the sending time, also indicated in figure 4.6, which suggests that channel competition can be reduced by spreading the sending point in time for the nodes.

Now, in figure 4.7, a custom made preemptive backoff strategy has been implemented. After the nodes get the instruction to start gathering samples and sending, they initialize by waiting a random time in the range 0 to $t_{sampling}$. Then they start record samples to fill a packet. Whenever a packet is full, given by the packet size used, they immediately send the packet, as earlier. In the illustration, node 1 initiates the sampling at time $t_{0,node1}$, node 2 starts at $t_{0,node2}$ and node 3 at $t_{0,node3}$. They are now spread out in time within one sampling period, and when the nodes are ready to send, there is no competition for the channel, and the chance of collision and dropped packets is greatly reduced. As the nodes start at a random point in time within the sample period, they may still start at the same time by chance, but on average they will start at different points in time.

The property discussed above is more effective with larger packet sizes, and enhances in importance as the number of nodes increases. A larger packet size means more samples per packet, resulting in a wider time window where the nodes may disperse, reducing the probability that two nodes try to access the channel at the same time.

4.1.4 Result Experiment 1 Revised

In figure 4.5, a preemptive backoff strategy has been implemented to relive the CSMA/CA algorithm. Other than that, the experiment is the same as the one presented in figure 4.4. Each point is still an average over five separate experiments with 60 seconds minute duration, and the same physical layout as before is used. The effect of the preemptive backoff strategy is dramatic. The packet loss is now reduced to less than 1% in nearly all test results. In more detail, the results are fractionally poorer for small packets than for medium sized and larger packets. Also, it seems that the number of retrys has little effect on the results. This implies that the backoff strategy had the desiered effect. Now that the nodes do not try to transmitt at the same time, the competition for the channel is lessened, and the need for resends is reduced. From this figure it is clear that the system can accomodate the three nodes and their data rates quite good. The errors that exist are presumably from transmission errors on the physical layer, i.e. multipath signals and fading. This picture gives a better indication as to what kind of errors that can be expected from the system, rather than the results without the preemptive backoff strategy implemented.

4.1.5 Result Experiment 2

So far, the testing has been performed on an interference "free" channel, i.e. IEEE 802.15.4 channel number 26, outside the IEEE 802.11 frequency range. To see what effect IEEE 802.11 has in terms of interference, the transmission was relocated to channel number 22. This channel has a center frequency of 2459 MHz (found from equation 2.1). The interfering WLAN was set to the IEEE 802.11b channel number 11, which has its

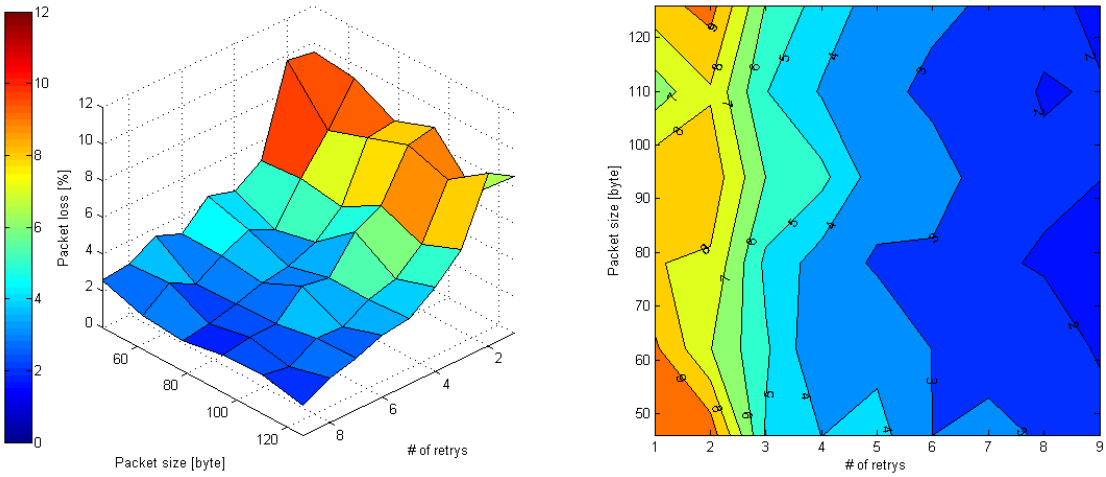


Figure 4.8: Test result with interference, standard backoff

center frequency at 2462 MHz, and spans for 11 MHz in both directions. The interfering signal strength in the vicinity of the nodes was measured by computer software to be approximately -40 dBm.

Figure 4.8 shows the packet loss for transmission without the custom preemptive backoff implemented using channel 22. The plot is very similar to the one for channel 26. The same tendencies of reduced packet loss for increasing number of resend possibilities is clearly visible. The minimum size packet gives the highest packet loss, and medium sized packets results in the best performance. The similarity to the interference free case suggests that the presence of a IEEE 802.11b network has minimal or no impact on the packet loss for the system being tested. The packet loss however, is still quite high.

4.1.6 Result Experiment 2 revised

Next, in figure 4.9, the same experiment is carried out, only this time with the preemptive backoff implemented. Again, the result speaks for itself. The improvement is radical, and the packet loss is greatly reduced. Now it is interesting to compare this result to the corresponding result in the interference free case, i.e. the result from channel 26 shown in figure 4.5. It is now more evident how the interfering WLAN transmitters effect the performance of the tested network. For the small packets, it results in an increase in packet loss of about 1% on average. However, the interference effects the experiments with a low number of possible retries for the most part. For larger number of allowed retires, the effect of the interference is reduced as one would expect. Another interresting result is that for the maximum size packets the result is also significantly worsened. In the interference free case, large packet size yielded a very low packet loss. With the

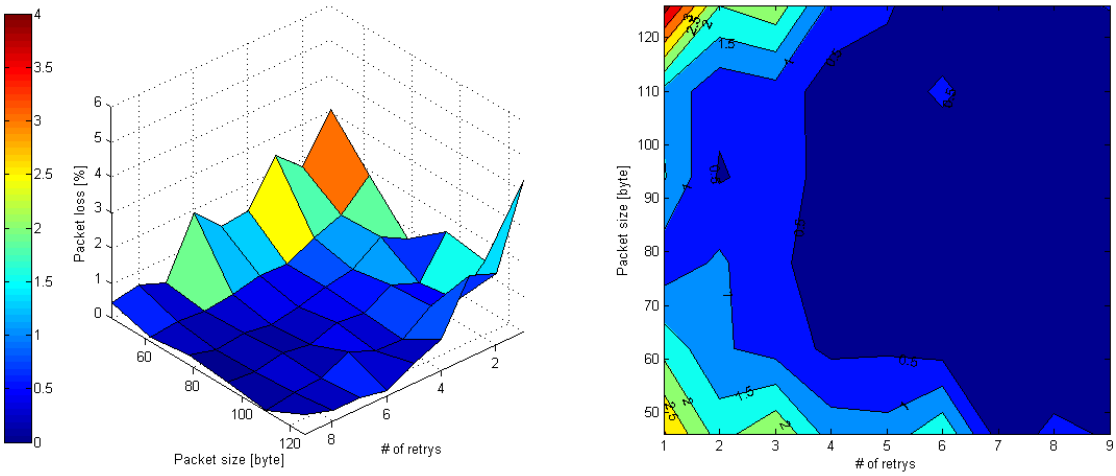


Figure 4.9: Test result with interference, preemptive backoff implemented

WLAN present, the maximum size packets gives rather high packet loss of about 3-4% for low retry count. For more retry possibilities, the packet loss flattens out towards a fractional percent.

4.1.7 Multi-hop Experiment

Figure 4.10 shows the results from the multi-hop experiment first described in 3.1.3 and its physical setup is illustrated in figure 3.2. The figure shows slightly improved performance with increasing packet size. It is worth noting the considerable variation in the results obtained at successive test runs. The results is not easily described by this figure alone, and to get a better picture of the results one should consider figure 4.10 together with 4.11. From figure 4.11 it comes clear that link quality is not the main reason for packet loss. One single node has two successive test runs with completely different results. The fact that nodes in one run has a zero loss performance and in the next testrun 75% packet loss points us to the conclusion that the motes can not handle the large amount of interrupts it has to handle. Since the node with the greatest packet loss changes with every run, it is not immediately clear which node dropped the packet.

4.1.8 Compression Results

The test results above imply that lager packet sizes are beneficial for the packet delivery success rate. In addition, the larger packets offer less overhead in the form of network infrastructure. Since the IEEE 802.15.4 standard defines an upper limit to the size of compatible packets, a further utilization factor based on the above results can

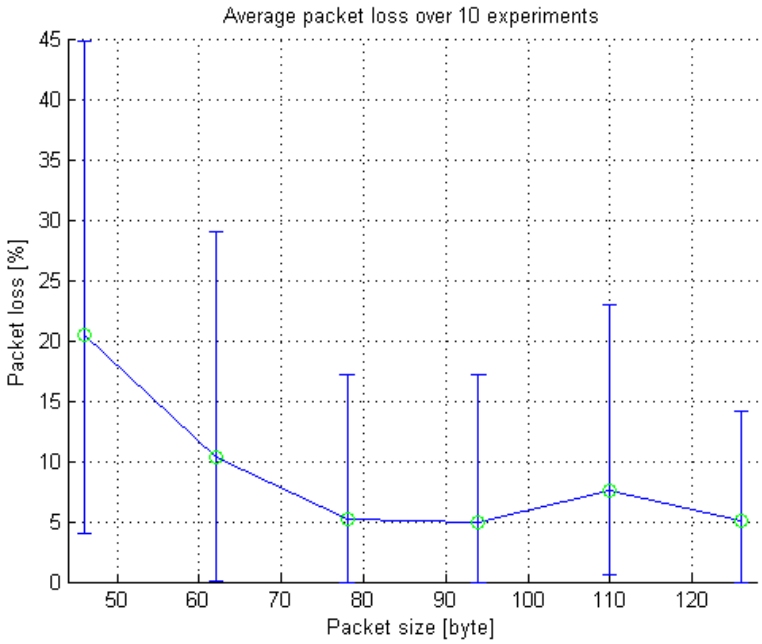
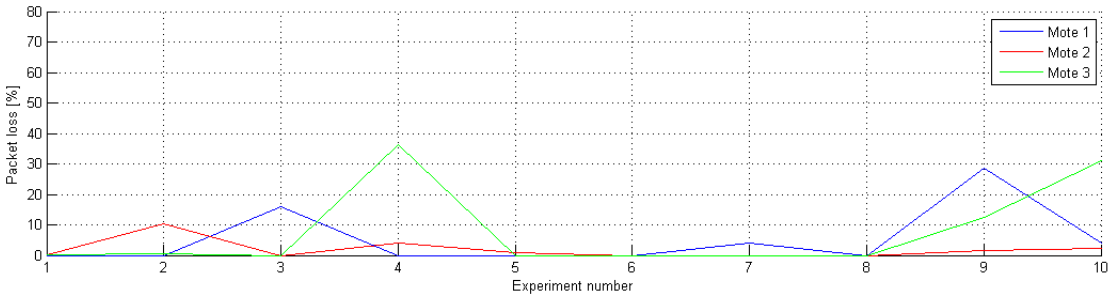


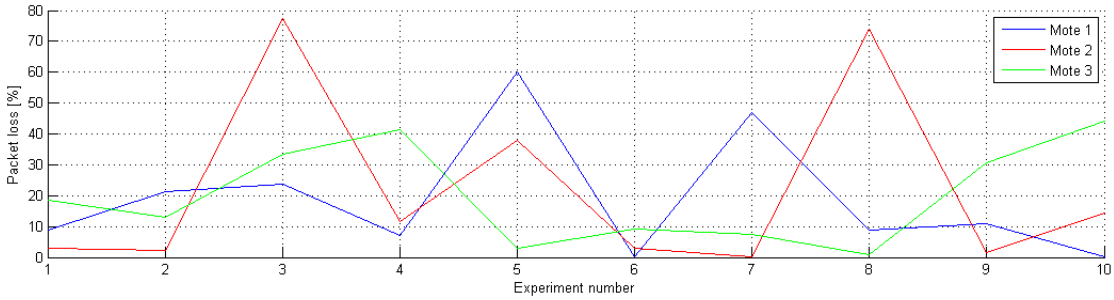
Figure 4.10: Multi-hop average test results

only be done by fitting more information into each maximum sized packet. With the implementation of the simple compression scheme described in chapter 3.2.8, the double amount of application data has been successfully transmitted without any alteration to the packet delivery success rate. The compressed message appends only one extra byte to the original packet and therefore does not change the properties of the channel transmission success rate gained from the earlier test results. The compression strategy is actually lossless. If a packet is lost, no error is propagated to successive packets or samples. The only effect is more than doubling of the network utilization, i.e. more samples in the same amount of bytes and less overhead per sample.

Figure 4.12 shows how the compression affects the effectiveness of the system. The two lower graphs show the obtained samples-per-byte ratio for the system without compression for single hop and multi-hop implementations respectively. Because the overhead in a single-hop system is 4 bytes less than in the multi-hop system, the single-hop performs slightly better. Without compression, each sample is represented by 2 byte and the overhead is 30 or 26 bytes for multi- and single-hop respectively. Thus, the maximum number of samples in a packet, with 127 bytes being the maximum allowed total packet size, is 48 as shown in the figure. When the delta encoding is implemented the effectiveness increases. In the plot, the two top graphs represent a system with 50% compression. Now each sample is represented by one byte after compression (except the first sample which is still 2 bytes for error control). It is also possible to put more samples in a packet



(a) 126 byte packages



(b) 48 byte packages

Figure 4.11: Results from 10 successive multi-hop test runs

within the allowed maximum packet size which increases sample-per-byte ratio.

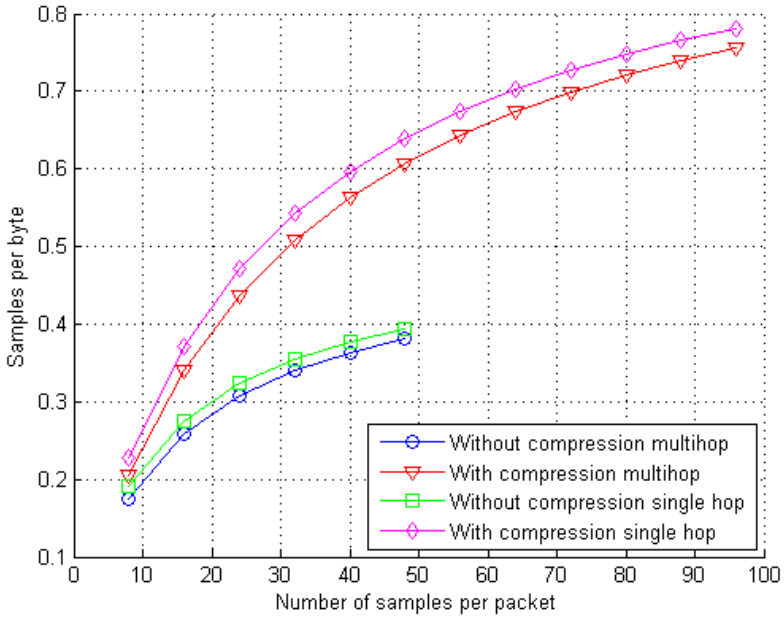


Figure 4.12: Effective number of samples per byte

4.2 Simulation Results

The first simulation result is shown in figure 4.13. It shows the packet loss for a network setup with three nodes and one controller. Several packet sizes and different number of possible retries were tested. Each point in the graph is a result of a 60 seconds simulation run as in the test described earlier in section 4.1.2. The simulation environment does not include any sources of interference in the 2.4 GHz band. In the figure, a steep decline in packet loss percent is clearly visible at 2 possible retries. This applies to all packet sizes used in the simulation. With more than 2 possible retries, the packet loss levels out at about 0%. For zero and one retry possibility, the packet loss is about 8%, and there is a slight tendency for higher packet loss with the minimal and maximal packet sizes. For the area above 2 possible retries, there are some variations, but the packet loss is always below 0.5%. The figure indicates that the three nodes have a tendency to have colliding transmissions attempts for less than 2 retries. When the number of possible retries is increased, the nodes have a wider time window to spread out in, and less packets are lost or discarded.

This simulation is done without any form of preemptive backoff scheme implemented. The shape of the figure is very similar to the one from the mote test shown in figure 4.4. In the hardware mote test, the packet loss is approximately 2% higher for all tests.

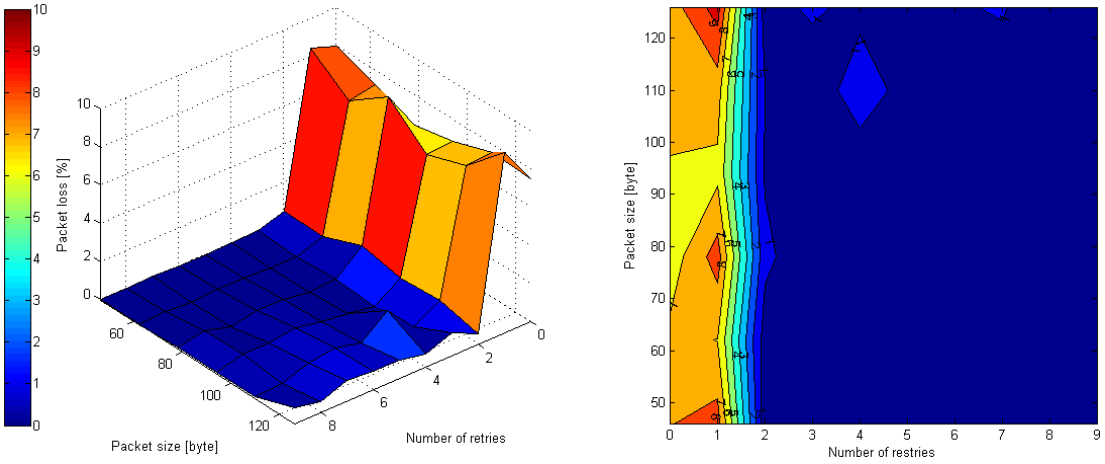


Figure 4.13: Simulation results for three nodes

The resemblance between the simulation and hardware test signifies the value of the simulations as an approximation to the real world system.

If the preemptive backoff strategy is implemented, the packet loss is reduced to zero for each and all packet sizes and simulated variations in the retry count. It is a remarkable reduction in packet loss for zero and one retry count from 8% to 0%. A comparison with figure 4.5 reveales that this simulation also corresponds quite good with the hardware test. In the simulation environment, a simple path-loss formula, only dependent of distance and a static factor, is used to calculate the received signal strength. In the real world test, fading and multiple reflected radio paths contribute to the error, which is not the case in the simulation.

The validity of the simulations have now been established. First, in figure 4.14, the multi-hop test described in section 4.1.7 is simulated to determine whether the network load of the three nodes in multi-hop is in excess of the capacity. The figure shows the simulation performed both with and without the custom backoff strategy. Without the backoff scheme, the results are varying and the packet loss is increasing with increasing packet size. However, when the custom backoff strategy is implemented, there is hardly any loss at all. This suggest that the standard is capable of handling the traffic from all three nodes through one node. A segment of the traffic produced in this simulation is shown in figure 4.15 for 46 bytes packet size. The upper line represents the channel access of node 4 (which in this case transmits ECG data at 200 Hz). The next line is the outgoing traffic from node 3 which now send both its own packets (ABP data at 100 Hz) and the data from node 4. The bottom line represents the traffic going from node 2 to the controller. It carries the data from all three nodes.

According to the calculations in section 2.1.5, only about 47 nodes can theoretically op-

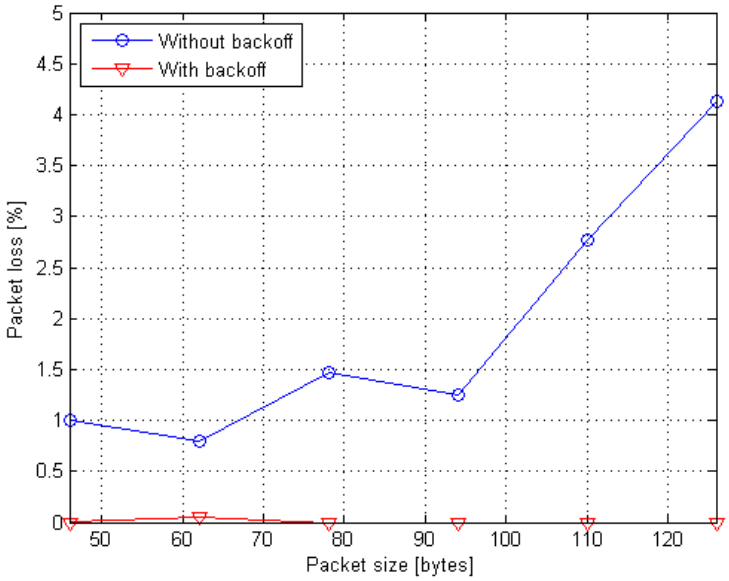


Figure 4.14: Simulation results for multi-hop test

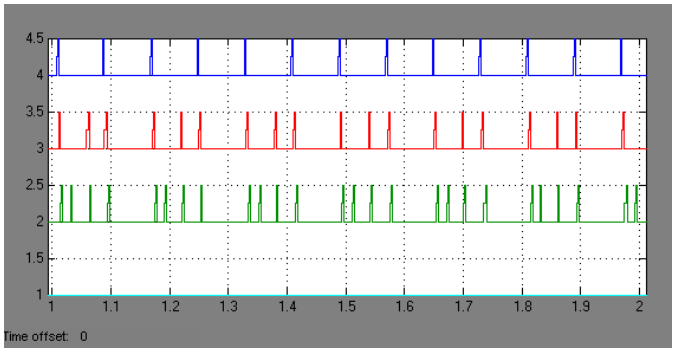


Figure 4.15: Network traffic multi-hop simulation

erate in the same network on the same channel with the given data rate. The simulation result shown in figure 4.16 shows the packet loss percent for a breakdown simulation. Different packet sizes are simulated for different number of nodes ranging from 3 to 45. All points are results of an average over three separate 30 seconds[1] simulations. The nodes were placed randomly within the radio coverage of the controller node, and different random positions were used for the three separate simulations.

[1]Only 30 seconds was simulated due to the computational complexity with a large number of nodes

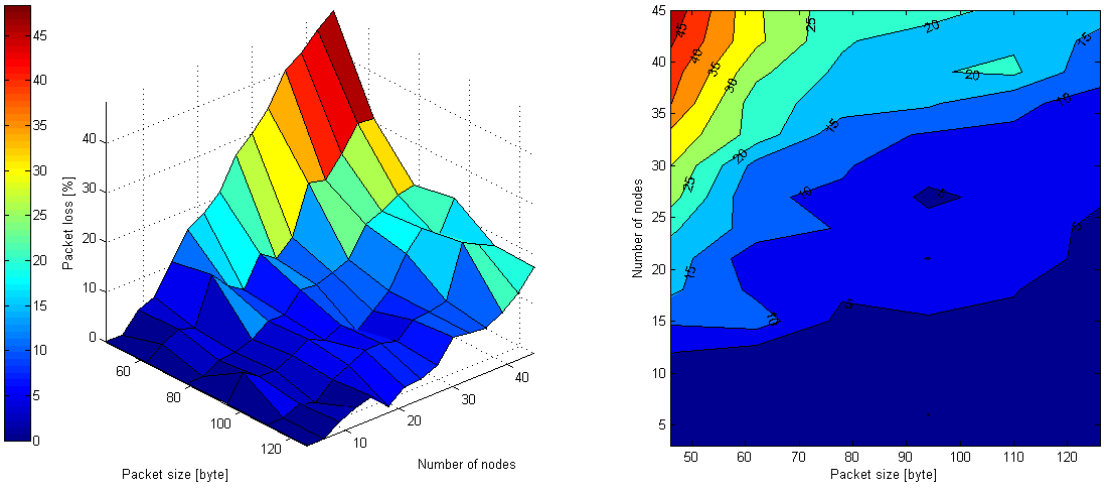


Figure 4.16: Simulation results for breakdown test

As the number of nodes increase, the packet error increases rather steady. Small packets saturate the network more than the large packets as was expected. With 45 nodes, the packet loss is about 45% for the minimal packet size and approximately 20% for the maximal packet size. The packet loss is below 5% up to about 10 nodes for small packets, and up to about 20 nodes for large packets. This is natural beacuse of the improved utilizaton of the network resources with large packets.

4.3 Summary of Results

Packet size	With preemptive backoff			Without preemptive backoff			Multihop
[bytes]	SIM	CH26	CH22	SIM	CH26	CH22	CH26
46	0,00 %	0,69 %	2,47 %	0,49 %	5,96 %	4,90 %	20,46 %
62	0,00 %	1,01 %	0,78 %	0,44 %	5,73 %	5,15 %	10,30 %
78	0,00 %	0,24 %	0,53 %	1,27 %	4,91 %	4,78 %	5,19 %
94	0,00 %	0,00 %	0,75 %	0,89 %	4,64 %	5,97 %	4,92 %
110	0,00 %	0,51 %	0,80 %	1,00 %	5,40 %	5,03 %	7,62 %
126	0,00 %	0,00 %	2,34 %	0,53 %	6,34 %	5,96 %	5,06 %

Table 4.1: Summary of experimental and simulation results

Table 4.1 summarizes the results from the experiments. The table is based on a possible retry count of 3 in view of the fact that this is the default number in the standard.

Chapter 5

Discussion

This chapter discusses some of the observations made from the experiments. In addition, some alternative techniques are suggested that might lead to improvements.

Link quality and coverage

There has been a general belief in the wireless sensor network community that the received signal strength indicator (RSSI) is a bad estimator of link quality. This belief is due to the existence of many asymmetry links in older radios such as CC1000 and TR1000, the choices of radio on the former MICA platforms. Newer radios based on the IEEE 802.15.4 standard, such as the CC2420 radio used on MICAz, implement another parameter called link quality indicator (LQI), which is believed to be a better indicator than RSSI. An extensive valuation of CC2420 is presented in [29] and concludes that hardware miscalibration is low or insignificant in CC2420 due to RSSI symmetry in links, and that RSSI is a promising indicator when its value is above the sensitivity threshold of CC2420 (-90dBm). The advantage that LQI has over RSSI is that it is much more chip-independent, i.e. hardware variations such as receive sensitivity are factored out. Based on these results, RSSI has been chosen as the basis for link quality estimation in this report. In the experiment illustrated by figure 4.1, the lowest registered value was -93 dBm. As expected, after hitting the -90 dBm threshold, the packages were received sporadic. We observed variations of -1 to -5 dBm in RSSI whenever the mote was moving, even though there was always line of sight between mote and base station. Although our experiments shows good quality links at distances up to 15 meters, the effective coverage distance is no more than 5 meters when mobility is accounted for. Moreover, the MICAz mote is also very sensitive to antenna *polarization*, and a 90 degree rotation of the antenna during the outdoor experiment resulted in a 7 to 8 dBm lower RSSI. This introduces a potential problem when it comes to designing wearable platforms.

RSSI has been proposed as a parameter for node *localization* and *tracing*. Our experi-

ments does not show a close relation between RSSI and distance. Indoor, if the motes are not close (distance $<$ 1 meter) the RSSI is not always a decreasing function of the distance, due to multipath fading and obstacles such as furniture. We were no closer to obtain a linear regression formula during the outdoor experiment (results shown in figure 4.2), and a general localization algorithm will probably fail due to this fact. This does not mean that RSSI could not be used for localization and tracking, but rather that there are several open problems related to this issue. In fact, [30] has proposed a localization algorithm based on trigonometric calculation of RSSI, that offers promising accuracy.

Inter node- and WLAN interference

During all experiments presented in this report, the motes were configured to transmit at full strength. Lower transmission power can be advantageous in order to reduce *interference* between nodes. In the scenarios described and valuated in this report, the nodes are placed relatively close and with a constant distance to each other. These nodes, which are intended for body area network, will be likely to have a relaying mote to forward data to the gateway. With the limited distance and density of nodes the transmission power used during testing has been held at a constant value. It would be difficult to choose an optimal transmission power for each separate node because of the small margins this would involve. In the case where several relaying nodes forward data from groups of nodes, power control becomes another issue. Groups of nodes forming clusters, with one relay node each, could evade much of the node interference by utilizing that the distance between neighboring clusters are significantly greater than the distance between cluster members. In addition, the network can from time to time become very dense with a large amount of nodes having a large degree of mobility. The network must therefore be able to handle very complex radio dynamics. With transmission power control the network density could in theory be confined within the coverage of the smaller sub-networks. Taking this even further, the relaying nodes could communicate on separate channels and thus reducing channel traffic. Though non-uniform transmission power would be beneficial for both energy consumption and reducing interference, the full exploration ratio of this property could not be confirmed with the limited number of motes available during our research. The TrueTime simulation environment does not support nun-uniform transmit power settings. Thus, the simulations could not be used to supplement the lack of experimental results.

In order to examine the significance of interference from the IEEE 802.11b/g wireless local area network (WLAN), experiments were carried out on two carefully selected channels. The channel with the least exposure to WLAN frequencies, and the channel closest to the most powerful WLAN center frequency. WLAN RSSI was measured to -40 dBm on the host computer. With little or no interference between WLAN and motes, the first experiment was carried out and monitored from the host computer. During the next experiment, involving overlapping frequencies, the host computer were

streaming over WLAN in addition to monitoring the nodes. The results from the two experiments imply that the effect of interference from WLAN is minimal. The results differs little more than one could expect from channel fluctuations over time. Since the two experiments could not be performed simultaneously we can only conclude that interference from WLAN has little effect on IEEE 802.15.4 in our experiments. However, we are aware that related research point in the opposite direction. Sikora and Groze show in [8] that for a worst case scenario with heavy WLAN traffic, approximately 90% of all IEEE 802.15.4 frames are destroyed by the interference. Part of the explenation for the differnt results may lie WLAN activity. Where we had a traffic load of about 128 kbps, Sikora and Groze operated with a load of approximately 3.4 Mbps.

Data rate optimization

As opposed to a majority of today's sensor networks, medical applications demand relatively high data rates and several mobile observers. As most standards, the intended applications has varying functionality and requirements. This property implies a general approach. IEEE 802.15.4 is intended for low power applications that supports "opportunistic ad hoc networking". Since this network should account for large amounts of nodes competing over the same channel there comes a trade-off between number of nodes, traffic load per node and QoS. As shown in chapter 4, the radio transmission success rate is highly influenced by the number of allowed retransmissions which the standard limits to 5. If the optional ACK is deployed, the effective end-to-end rate will also depend on the overhead from transmitting the ACK packet. Figure 4.4 and figure 4.8 clearly show the effect of this overhead. Here the test results with no ACK shows better results than the test results with ACK and one retransmission enabled. Further the results shows an improvement in packet delivery success as the number of allowed retransmissions increase. Our results show increasing gain even at 9 retransmissions. The result may look completely different in an environment with emphasis on either of the other trade-offs, i.e. network density or QoS. In view of the wide diversity of IEEE 802.15.4 applications it is clear that great improvements can be made upon the standard at higher layers. One requirement of medical applications is QoS which is not prioritized by the standard. To fulfill the requirement of QoS, additional retransmissions should be enabled on the higher levels. Even though the random backoff time increases the transmission success rate greatly, an additional preemptive backoff time would relieve the standards CSMA/CA algorithm, and offer improvement to networks with periodic transmission and especially applications with high data rates and data buffering intervals. Results presented in this report show a dramatic effect of this particular strategy. Another important observation from our experiment is the packet size. The default packet size in TinyOS has 29 bytes of payload, which sums up to a total packet size of 41 bytes on the MICAz mote. The results presented in chapter 4 questions the rationale for this choice of packet size. The reason for this particular choice is most likely historical rather than practical. This is confirmed by one of the developers in [31]. The original

TinyOS motes had a radio that was controlled by software at the bit level, and the developers found that 36 bytes/packet was a suitable value to maximize throughput, and minimize chance of packets getting lost because of software timing glitches. The smaller packet size avoids the message buffers taking too much RAM. Furthermore bandwidth is not usually the critical metric for low-power embedded wireless nodes that have a <1% duty cycle.

However, our results show that the optimal payload size depends on what type of environment the network exists within. With any radio, by increasing the available data length, the effective packet header:data ratio is decreased. Increasing the data length as much as possible, would in ideal cases increase the data throughput to its maximum. This is because it takes less header information per data information to transmit, and because the IEEE 802.15.4 radio requires some time to perform the CSMA/CA algorithm. In the worst case, with default configuration and heavy load on the channel, the time between successive transmissions derived from equation 5.1 and IEEE 802.15.4 defined values is 134,4 ms. So by decreasing the number of transmitted packets while still getting the same amount of data through, the throughput is obviously going to increase.

$$\begin{aligned} T_{send} &= (((2^{BE_{\#macCSMABackoff}} - 1) * T_{aUnitBackoffPerids} + T_{CCA}) \\ &\quad + macAckWaitDuration)_{T_{\#frameRetry}} \end{aligned} \tag{5.1}$$

If the network is in a noisy environment, then there is an increased chance for the packet to get corrupted during transmission. In this case, it will take longer to recover that packet because of its length, and instead of losing only a little bit of data, a lot of data is lost. The obvious question is then; what is the nominal packet length that minimizes the effects of noise while maximizing throughput? Though the results motivate the use of the maximum packet size, the question still remains a design problem. Indoors with tight spacing, maximizing the payload would be beneficial, whilst outdoors on the ground with wider spacing, the payload size might as well be kept pretty modest.

Simulations are done for transmission with varying packet sizes, but it is difficult to port these results over to the real world because the packet success rate in a network environment may change over time, even by the minute as illustrated by figure 4.3. As both the simulation and the test results imply, the packet payload length should be increased to the maximum 127 bytes, assuming the number of message buffers do not exceed the free RAM on the mote needed to execute the application.

The above discussion is valid but inadequate for multi node networks, where several motes are competing for the channel access. In such networks the traffic load, given by the amount of packets transmitted within any given time frame, becomes the most important factor. While considering the possibility of packets colliding on the channel during transmission it becomes clear that performance will improve with the reduction of traffic load. As verified in chapter 4, three ways of traffic load reduction is possible;

increasing packet size, compressing the data or a combination of the two. Again the results imply the use of maximizing packet payload. The drawback with the use of this practice is the signal delay introduced by the extra buffering of sampled data, the additional memory usage and program complexity. In the results from the breakdown experiment shown in 4.16 the packet loss is approximately proportional to the number of nodes, i.e. for a distinct packet size, the packet loss increases with a constant slope with increasing number of nodes. For the minimal packet size, the proportionality constant is about 1, whereas the maximum packet size has a proportionality constant of roughly 0.5. From figure 4.16 it is possible to derive the maximum number of nodes that can be used for different packet sizes given a upper bound for packet loss. The number of nodes that can be supported within the network below a 5% error threshold is 10 to 20 nodes depending on the packet size. 10 to 20 nodes is what one could expect to be the node density for a confined area in medical applications. This result corresponds well with the theoretical bound for node density derived in 2.1.5.

The compression scheme presented in this report (2.2, 3.2.8, 4.1.8) shows promising results in spite of its simplicity. When it comes to trading energy consumption against greater complexity the simple delta coder performs well. While bisecting the application data before transmission, the added complexity is at a minimum. No extra buffering is needed and computing the derivate of the original signal is run as a task after all the samples are ready, and will not cause any hardware interrupts. Further more, the overhead in application size is a minimum. Since all valid packets must pass the CRC criterion, the compression does not induce any increase in packet error. The additional delay time as a consequence of larger packet sizes is not really significant as the maximum packet size supported by IEEE 802.15.4 is relatively small. By increasing the packet size from eight sample payload to 48 samples payload in in the maximum sized packets, the delay increases from 40 to 240 ms, which is still an acceptable delay.

Multi-hop evaluation

Multi-hop networks introduce numerous advantages and possibilities. Wireless sensor networks intended for medical applications might be deployed both in and outside of hospitals. Outside of the hospital one could in most cases not guarantee the existence of infrastructure assisting the organization of nodes, and the nodes need to organize themselves [32][33]. The multi-hop experiment described first in section 3.1.3 and later in section 4.1.7, were performed to verify whether the nodes could handle the forwarding of application data over several hops. A second goal was to test some simple routing algorithms. The end result was node starvation with the simplest routing protocol we could think of. In this routing algorithm, every node sends its packages to its local address minus one, i.e. after four hops node 4 would reach the base station at address 0. A possible explanation for the poor multi-hop performance can be given by the fact that every node is in range of each other as illustrated by figure 3.10. This is the case for both the single-hop and the multi-hop experiment. The difference is the

incresed number of packages caused by forwarding and the not so intuitive fact that packages addressed to a node with a given coverage range causes an interrupt with all other nodes also within this range. Because the mote will have to examine the package to check *whether* it was intended for itself, the increase in traffic load does not just affect the intermediate motes but all motes within range of the source. Take for example the experiment with star network topology. Three motes transmits 12, 12 and 24 packets per minute. Thats 48 packets received at the base station every minute. In the multi-hop experiment the motes transmit the same amount of packages but trough intersecting motes. The motes now transmit 24, 32 and 48 packets per second respectively. But since they are all within each others transmission range, the base station and every other mote have to examine 104 packets per minute in stead of 48. As the number of nodes increase, the traffic load increases exponentially and the large amount of hardware interrupts causes a network breakdown. This result was perhaps the biggest disappointment during experimentation and we were forced to drop the planned routing experiments. This is a big setback given that functional multi-hop protocols should, in addition to relay application data, manage routing tables with several alternative routes, manage data aggregation and exchange local state information and meta data on regular basis. The TinyOS library contains low complexity routing protocols, as according to the application's documentation supports a maximum of one message every other second. Within this time frame, our application needs to relay 8 packets, given the maximum packet size, in addition to uphold infrastructure. Our results supports what was concluded in [34], that motes and TinyOS are insufficient for running the MAC layer in software, and in particular insufficient for running more than the needed communication. Thomsen et al. could not manage to run either MICAz or the slightly better equipped TELOS rev.A in beacon mode as a full function device (FFD). The simulation however, shows zero packet loss for the equivalent experiment. In the simulation, the motes have infinite processing capacity. Thus, simulation supports the theory that the micro controller has reached its limits regarding capacity.

Latency

Communication should happen with low latency, so that vital changes in the patient's medical condition can be observed immediately, for example if the system should function during an operation. There are mainly three factors affecting the throughput delay time. The motes are configured to wait for n samples before a full packet is ready for transmission, i.e a delay of

$$\frac{n}{F_{sample}} \; seconds$$

In multi-hop networks, the end-to-end latency is also affected by the number of interleaving nodes. Every time the package is received by a interleaving node, the CSMA/CA algorithm is run, it adds to the total delay time (revisit equation 5.1). The third parameter in total delay time includes compression and data aggregation. The actual time

required to fill a packet depends on the number of samples in each packet. While sampling with 200 Hz and 16 bit quantizer, the required time to fill a maximum sized packet is 5 $ms * \#samples \approx 240\ ms$, or 480 ms for 50% compressed packets, which is an acceptable delay. The real trade off here is the acceptable packet delivery success rate, i.e. if a packet does not reach its intended destination, how long should the mote try to get the packet trough before dropping it. The IEEE 802.15.4 standard does not offer quality of service (QoS) and packets are therefor sent on best effort. Because the time required to fill the packet is approximately 50-100 times greater than the required channel access time, the motes could queue a number of messages for transmission upon clear channel. There is however little available RAM on the motes and in a real time scenario the data becomes less valuable with the passing of time. It is very difficult to guarantee packet delivery on one side and on the other side guarantee that the packet is not older than a pre-defined value. By dropping expired packets at the source, one could at least guarantee that all received packets are relatively fresh, depending on the number of retransmissions required by intervening nodes.

Chapter 6

Conclusion and Future Work

By implementing our own sensor network application with IEEE 802.15.4/ZigBee support we experimented with actual medical data such as arterial blood pressure, central veinous pressure and electrocardiogram. We focused on computer simulations and hardware testing of different scenarios, like in operating rooms and intensive care units, in order to employ a realistic approach. Several issues have been addressed such as Zigbee/WLAN coexistence, coverage and multi-hop properties. By utilizing typical biomedical properties, techniques including compression and preemptive collision avoidance was implemented and shown to have a dramatic effect on the transmission success rate.

Data gathering from stationary motes, deployed with star topology, shows packet loss below 1% for three motes communicating with individual data rates 1.6 and 3.2 kbps. Simulation results also show that the packet loss is below 5% for 10 nodes given a small packet size, and just above 20 nodes for the larger packets. Experiments executed with moderate interference from WLAN did not result in significant additional packet loss, 1-4 % depending on packet size. So far the results are promising. Routing data though intermediate motes exponentially increases traffic load given that all the motes are within the same coverage area. This puts the hardware to its limit and mote starvation is the result with only four motes including one mote configured as a base station. The simulation however, shows zero packet loss for the equivalent experiment. This points us to the conclusion that the current mote and TinyOS can not handle the requirements for a multi-hop medical sensor network. However, by adapting the standard to the data requirements and implementing compression, considerably better performance is achieved. Future work should include large scale testing and more realistic channel model for the simulation. Additional compression could be performed to further approach the entropi of the input signals. The current compression scheme could be extended to a DPCM coder, and experimentation with variable length sample representation could also result in improved performance. This would involve added complexity to the application, and a survey on energy consumption should be performed in order to verify the compression algorithm's impact on system energy consumption. Further work should also emphasize

on energy consumption in order to prolong network life span. Mote mobility has an impact on link stability and needs to be addressed. The slotted beacon mode holds high expectations and needs to be evaluated once sufficient hardware and software support becomes available. There are still many unsolved and unaddressed issues regarding medical sensor networks with IEEE 802.15.4. The MICAz platform is to immature for heavy network operations like routing of large amounts of data. The full potential of IEEE 802.15.4 is yet to be investigated with the beacon mode. Energy and security issues have to be addressed together with QoS. All in all, the results from the experiments point in a negative direction, but with improvements to the platform and OS together with a thought-through adaption of the standard, acceptable results may emerge.

Bibliography

[1] The MobiHealth Project. Innovative GPRS/UMTS Mobile Services for Applications in Healthcare. http://www.mobihealth.org/.

[2] Smart: Scalable medical alert and response technology, June 2006. http://smart.csail.mit.edu/.

[3] Bruno Bougard, Francky Catthoor, Denis C. Daly, Anantha Chandrakasan, and Wim Dehaene. Energy Efficiency of the IEEE 802.15.4 Standard in Dense Wireless Microsensor Networks: Modeling and Improvement Perspectives. *Design, Automation and Test in Europe (DATE'05)*, Vol.1:196–201, 2005.

[4] Mats Skogholt Hansen. A comparative study of wlan, bluetooth and zigbee for medical bio-sensor networks. Project at department of Electronics and Telecommunications, NTNU, December 2005.

[5] Zigbee Alliance Specification, May 2006. http://www.zigbee.org/.

[6] IEEE Standard for Information Technology. Part 802.15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs). 2003.

[7] Tony Sun, Ling-Jyh Chan, Chih-Chieh Han, Guang Yang, and Mario Gerla. Measuring Effective Capacity of IEEE 802.15.4 Beaconless Mode, 2006. http://netlab.cs.ucla.edu/wiki/files/TS2006WCNC.pdf.

[8] Axel Sikora and Voicu F. Groza. Coexistence of IEEE 802.15.4 with other Systems in the 2.4 GHz-ISM-Band. In *Instrumentation and Measurement Technology Conference, IMTC 2005, Proceedings of the IEEE*, 2005.

[9] Soo Young Shin, Sunghyun Choi, Hong Seong Park, and Wook Hyun Kwon. Packet Error Rate Analysis of IEEE 802.15.4 under IEEE 802.11b Interference. In *Proc. Conference on Wired/Wireless Internet Communications (WWIC'2005)*, 2005.

[10] Simon Haykin. *Communication Systems*. John Wiley & Sons, Inc., 4 edition, 2001.

[11] Karl Øyri, Ilangko Balasingham, Eigil Samset, Jan Olav Høgetveit, and Erik Fosse. Wireless Continuous Arterial Blood Pressure Monitoring During Surgery: A Pilot Study. In *Anesth. Analg. 2006 102: 478-483.*, 2006.

[12] Andrew S. Tannenbaum. *Computer Networks*. Prentice Hall PTR, 4 edition, 2003.

[13] Jerry D. Gibson, Toby Berger, Tom Lookabaugh, Dave Lindbergh, and Richard L. Baker. *Digital Compression for Multimedia, Principles and Standards*. Academicv Press, 1998.

[14] Crossbow Technology inc. MicaZ ZigBee Series (MPR2400), June 2006. http://www.xbow.com/Products/productsdetails.aspx?sid=101.

[15] TinyOS Hardware Designs, June 2006. http://www.tinyos.net/scoop/special/hardware.

[16] TinyOS Community Forum, May 2006. http://www.tinyos.net/.

[17] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC Language: A Holistic Approach to Network Embedded Systems. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI)*, 2003.

[18] The Contiki Operating System, May 2006. http://www.sics.se/~adam/contiki/.

[19] SOS Operating System, May 2006. http://nesl.ee.ucla.edu/projects/SOS/publications/.

[20] MANTIS Project, May 2006. http://mantis.cs.colorado.edu/.

[21] Philip Levis. TinyOS Programming, February 2006. http://csl.stanford.edu/~pal/pubs/tinyos-programming-1-0.pdf.

[22] Chipcon AS. CC2420 Data Sheet, October 2005. http://www.chipcon.com.

[23] TrueTime: Simulation of Networked and Embedded Control Systems, March 2006. http://www.control.lth.se/~dan/truetime/.

[24] OPNET Technologies, Inc, January 2006. http://www.opnet.com/.

[25] National Institute of Standards and Technology, NIST, January 2006. http://w3.antd.nist.gov/Health.shtml.

[26] OMNeT++ Community Site, February 2006. http://www.omnetpp.org/.

[27] NesCT: A Language Translator, February 2006. http://nesct.sourceforge.net/.

[28] Martin Andersson, Dan Henriksson, and Anton Cervin. TrueTime 1.3–Reference Manual, 2005. Department of Automatic Control, Lund Institute of Technology.

[29] Kannan Srinivasan and Phillip Levis. RSSI is Under Appreciated. In *Proceedings of the Third Workshop on Embedded Networked Sensors (EmNets 2006)*, To appear.

[30] Konrad Lorincz and Matt Welsh. MoteTrack: A Robust, Decentralized Approach to RF-Based Location Tracking. In *Proceedings of the International Workshop on Location and Context-Awareness (LoCA 2005)*, Mai 2005.

[31] TinyOS-help Archive, May 2006. http://mail.millennium.berkeley.edu/pipermail/tinyos-help/2006-April/016191.html.

[32] Stig Støa. Sensornettverk for medisinsk behandling. Project at department of Electronics and Telecommunications, NTNU, December 2005.

[33] The CodeBlue Project. Codeblue: Wireless Sensor Networks for Medical Care, June 2006. http://www.eecs.harvard.edu/~mdw/proj/codeblue/.

[34] Jonas Thomsen and Dirk Husemann. Evaluating the use of motes and tinyos for a mobile sensor platform. In *Proceedngs of the 24th IASTED International Multi-Conference parallell and distributed computing and networks*, February 14-16 2006.

Appendix A

AM Message Structures

The MicaZ IEEE 802.15.4 compatible message structure is defined by the struct TOS_Msg in the file \tos\platform\micaz\AM.h. This data structure is defined as follows:

```
typedef struct TOS_Msg
{
  /* The following fields are transmitted/received on the radio. */
  uint8_t length;                    // Payload length
  uint8_t fcfhi;                     // Frame control
  uint8_t fcflo;                     // Frame control
  uint8_t dsn;                       // Sequence number
  uint16_t destpan;                  // Destination PAN ID
  uint16_t addr;                     // TOS address
  uint8_t type;                      // TOS AM type
  uint8_t group;                     // TOS group ID
  int8_t data[TOSH_DATA_LENGTH];     // MAC service data unit

  /* The following fields are not actually transmitted or received
   * on the radio! They are used for internal accounting only.
   * The reason they are in this structure is that the AM interface
   * requires them to be part of the TOS_Msg that is passed to
   * send/receive operations.
   */
  uint8_t strength;                  // RSSI value provided by radio
  uint8_t lqi;                       // LQI value provided by radio
  bool crc;                          // MAC frame check sum
  uint8_t ack;                       // ACK required / not required
  uint16_t time;                     // Time stamp
} TOS_Msg;
```

For non- IEEE 802.4.15 compatible motes, the payload data will typically be a type of TinyOS message, as defined by the struct TOS_Msg in the file \tos\types\AM.h. This data structure is defined as follows:

```
#define TOSH_DATA_LENGTH 29  //default payload length

typedef struct TOS_Msg
{
  /* The following fields are transmitted/received on the radio. */
  uint16_t addr;
  uint8_t type;
  uint8_t group;
  uint8_t length;
  int8_t data[TOSH_DATA_LENGTH];
  uint16_t crc;

  /* The following fields are not actually transmitted or received
   * on the radio! They are used for internal accounting only.
   * The reason they are in this structure is that the AM interface
   * requires them to be part of the TOS_Msg that is passed to
   * send/receive operations.
   */
  uint16_t strength;
  uint8_t ack;
  uint16_t time;
  uint8_t sendSecurityMode;
  uint8_t receiveSecurityMode;
} TOS_Msg;
```

Appendix B

Message Injection Application

Every request by the test application is injected into the sensor network by making a call with the following semantics:

```
ZBcastInject <command> [argument]
           <led_on>
           <led_off>
           <channel> [Radio Channel ]
           <TXPower> [Transmit Power]
           <read_log>  [dest_address]
           <write_log> [dest_address] [data_to_log] [data_to_log] ..
           <log_file>  [dest_address] [ data_file ]
           <start_sensing> [#samples] [interval ms] [#retries]
```

Where

channel resets the current active channel. Argument is the channel number given by equation 2.1 (11-26).

TXPower resets the motes current transmission power. Valid aguments are listed in table 2.2 which also shows the corresponding dBm values.

read_log returns a LogMsg message. The command requires a mote ID as an argument.

write_log writes up to 16 bytes to the EEPROM. Arguments are destination mote ID and signed integer sample values.

log_file moves a data file from the host computer over the radio and into the re-mote's EEPROM. Data file must be a .dat file with signed integers separated by new line.

start_sensing requests that the re-motes start sampling data and transmitting when their buffers are full. Arguments are the number of 16 byte samples arrays requested, the sampling intervall in ms, and the maximum number of retries on transmission failure. If all arguments have value 0, the re-motes try to find correct values from the local logfile.

ZBcastInject <command >[argument] generates the following message:

```
typedef struct ZigmedCmdMsg {
    int8_t seqno;                // For broadcast message comparison
    int8_t action;               // Command to schedule
    uint16_t source;             // Original source address
    uint8_t hop_count;           // Rebroadcast counter
    union {
      start_sense_args ss_args;  // Arguments for sensor component
      read_log_args rl_args;     // Broadcast message intended for specific mote
      write_log_args wl_args;    // Values to append log
      uint8_t untyped_args[0];   // Enables variable size message in MIG
    } args;
} ZigmedCmdMsg;

/*
 * UNION FIELD:
 */
typedef struct {
    int nsamples;       // Request ''nsamples'' lines from the log
    uint32_t interval;  // Read one line every ''interval'' ms
    uint8_t radioconf;  // Number of allowable retransmissions
} start_sense_args;

typedef struct {
    uint16_t destaddr;  // Broadcast message intended for specific mote
} read_log_args;

typedef struct {
  uint16_t destaddr;  // Broadcast message intended for specific mote
    uint16_t wlog[8];   // 8bit x 16 = one line in EEPROM (16 lines = one page.)
} write_log_args;
```

Appendix C

The CSMA/CA Algorithm in IEEE 802.15.4

C.1 Unslotted CSMA/CA

When in unslotted mode, firstly variables NB and BE are initiated. NB, the number of backoffs completed is set to zero and BE, the backoff exponent is set to macMinBE which is user defined within the interval 0 to 5. Next, a number of unit backoff periods is found as a random number in the interval 0 to $2^{BE} - 1$, and nothing is done within this time interval. Then a Clear Channel Assessment is performed to sense if the channel is busy. If the channel is idle the node sends a packet on the channel. If the channel is sensed to be busy, NB is set to NB+1, and BE is set to BE+1 where the maximum value of BE is 5. If the number of completed backoffs NB is less than a maximum allowed number (3 to 5, user defined), if goes back to finding a new delay period. If the maximum number of backoffs has been completed, a transmission failure is reported.

C.2 Slotted CSMA/CA

The big difference in the slotted mode is that the channel sensing is performed at defined backoff period boundaries. In addition, a Contention Window CW is defined. The CW is set to 2 prior to a new backoff execution. In practice this means that all devices have to sense the channel twice before accessing the channel. In slotted mode a battery life extension mode is also supported. This puts a restriction on the BE, which in turn makes the possible waiting time to access the channel shorter. As in the unslotted mode, a failure is reported if the channel is sensed busy a maximum allowed times.

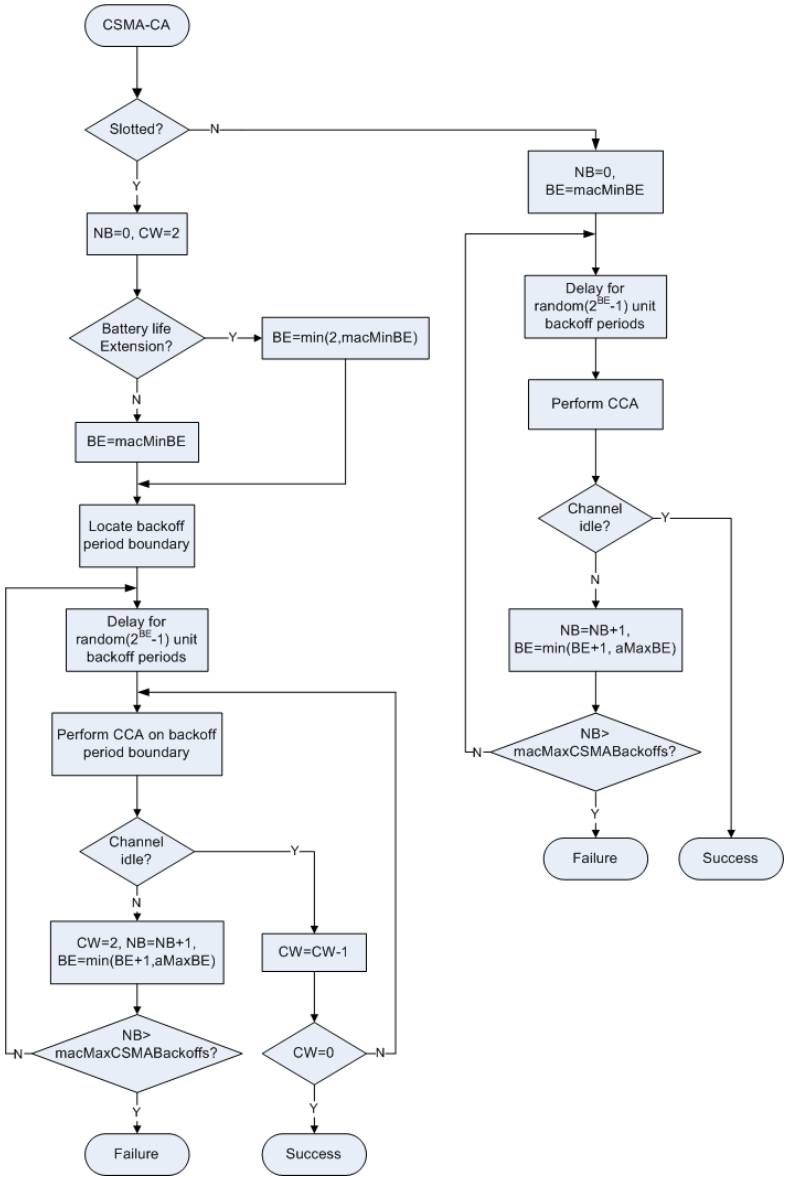


Figure C.1: The CSMA-CA algorithm in ZigBee

Appendix D

Source Code Example, Delta Coder Component

```
module DeltaCoderC{
   provides interface DeltaCoder;
}
implementation{
   uint8_t iterator, BUFFERSIZE;;
   signed int currentValue, nextCurrentValue;
   signed char *bufOut;
   signed int *data;

   command result_t DeltaCoder.delta( uint8_t *bufferOut, uint16_t *bufferIn, uint8_t size ){
      BUFFERSIZE = size;
      data = bufferIn;              /*  use first buffer and make buffer signed  */
      bufOut = bufferOut;

      currentValue = *data;         /*  compress first buffer  */
      nextCurrentValue = *(++data);
      *(&bufOut[2]) = currentValue - nextCurrentValue;
      for (iterator = 3; iterator <= BUFFERSIZE; iterator++ ){
         *(&bufOut[iterator]) = *data - *(++data);
      }
      currentValue  = *(data);     /*  keep last value for append  */
      return SUCCESS;
  }

  command result_t DeltaCoder.append( uint16_t *bufferIn ){
     data = bufferIn;              /*   use second buffer  */

     *(&bufOut[BUFFERSIZE+1]) = currentValue - *data;  /*  compress second buffer  */
     for (iterator = (BUFFERSIZE+2); iterator <= (2*BUFFERSIZE); iterator++ ){
        *(&bufOut[iterator]) = *data - *(++data);
     }
     return SUCCESS;
  }
} // END imlementation
```

Appendix E

Mote Application Diagram

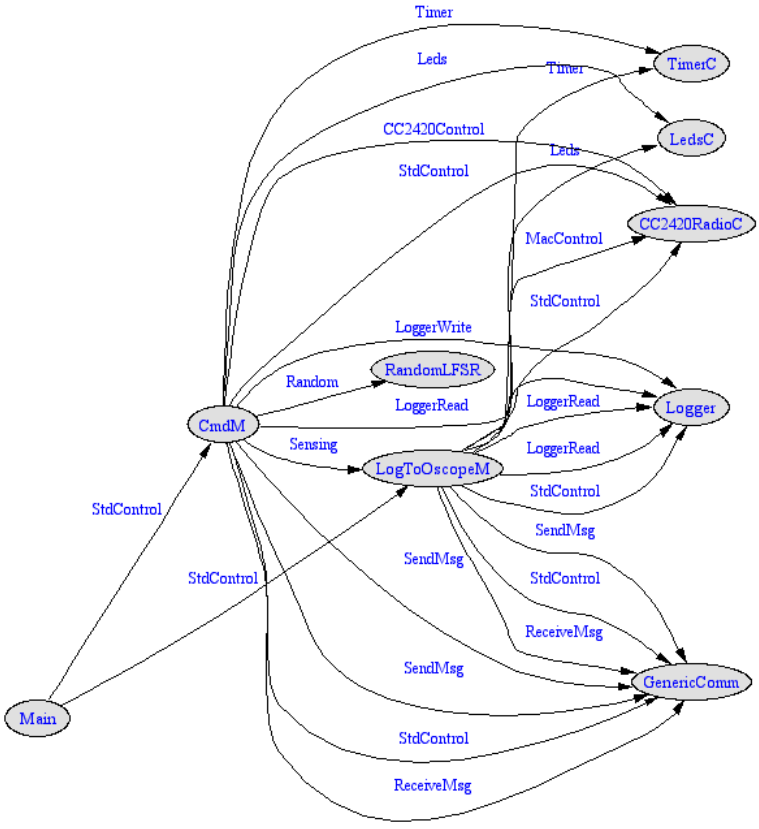


Figure E.1: Mote application diagram

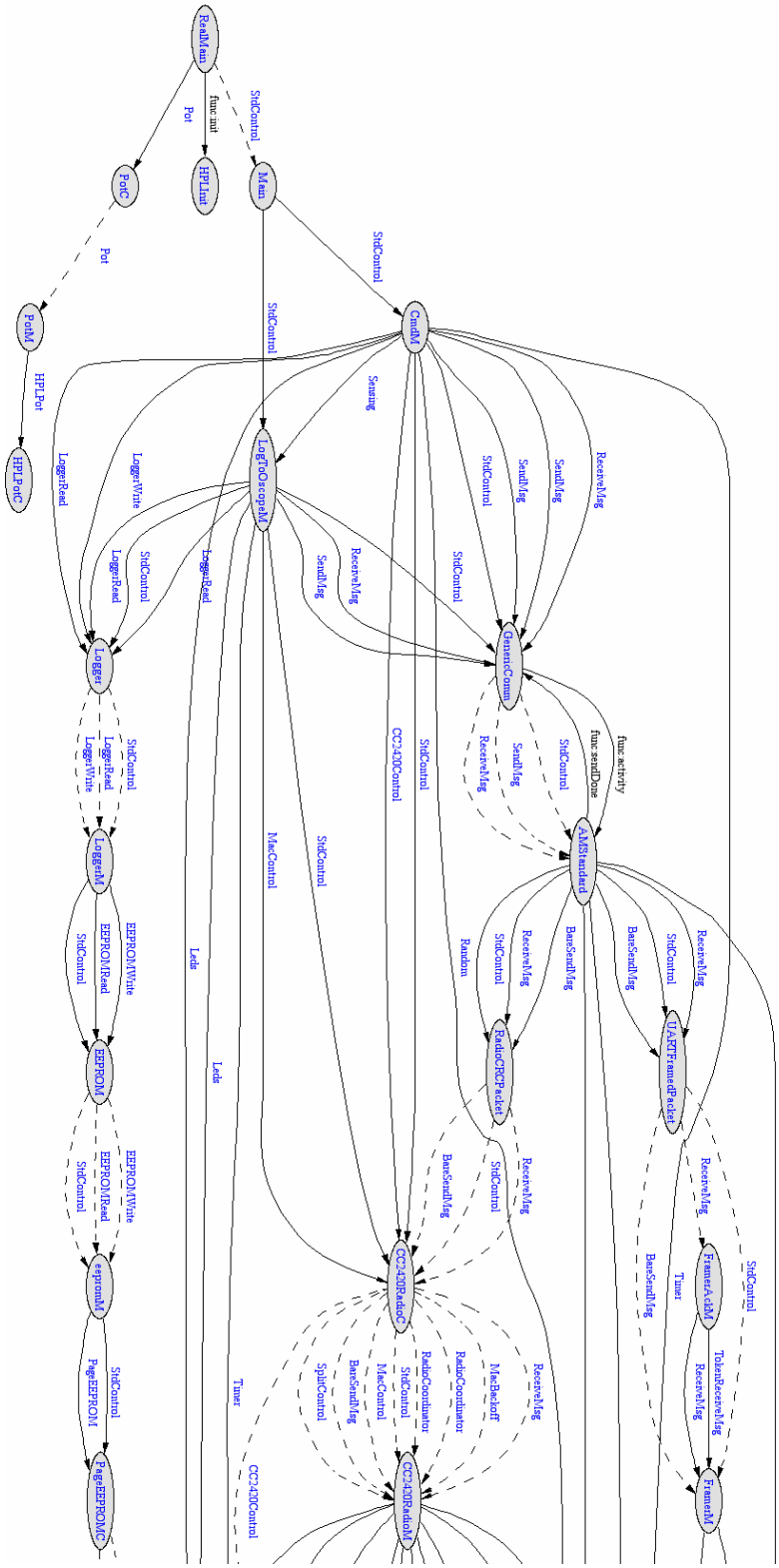


Figure E.2: Mote OS diagram chart

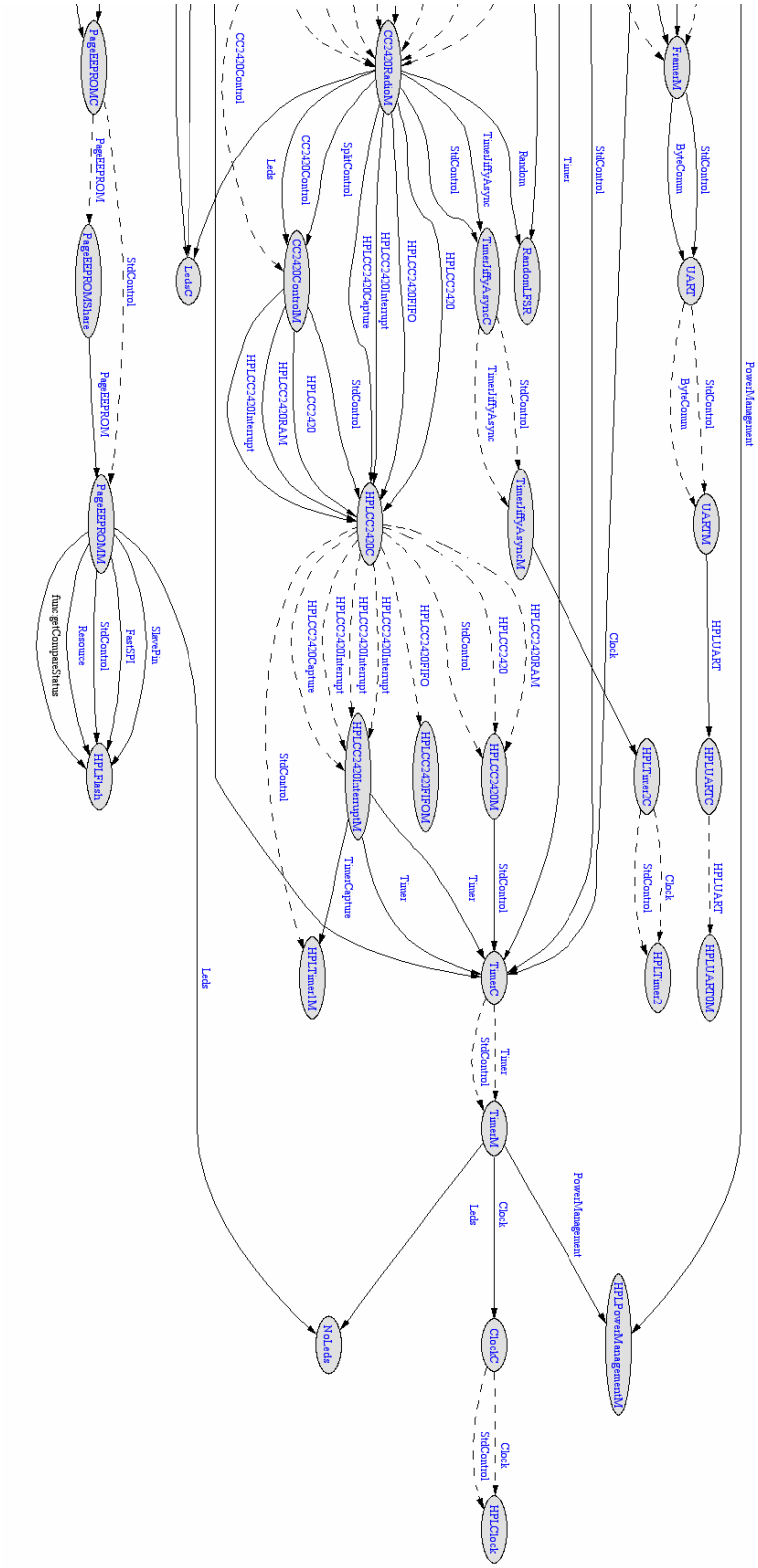


Figure E.3: Mote OS diagram (contd.)