NTNU
Innovation and Creativity

An Optimized Cross-Layer Protocol For Patient Confined Wireless Network

Loc Tan Vo

Master of Science in Communication Technology

Submission date:	July 2007
Supervisor:	Ilangko Balasingham, IET
Co-supervisor:	Stig Støa, Rikshospitalet

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

A body area network may consist of several sensors measuring different physiological markers. A wireless body area network will facilitate patient mobility. In this master thesis a customized and tailor-made communication protocol for hospital and home environments will be studied. Taking into account of network reliability and network robustness, data throughput optimization and practical implementation of the system will be performed.
The research platform will be based on T-Mote Sky, using the IEEE 802.15.4 PHY and MAC layer in a ZigBee network.
Issues on base station bandwidth and coverage area will be investigated.
The performance of the designed protocol in different realistic medical scenarios, with and without mobility, should be tested and provided. The report should contain description of the scenarios, simulation models, simulation results and discussions. Comparisons of results to similar existing implementations will be beneficial.

Assignment given: 25. January 2007
Supervisor: Ilangko Balasingham, IET

Preface

This Master thesis summarizes the work done in the 10th and last semester at the Master's program at the Department of Electronics and Telecommunications at NTNU. The project was initiated by Ilangko Balasingham at the National Hospital of Norway, and the work has partly been carried out at NTNU and partly at the National Hospital of Norway with Balasingham as head supervisor. Stig Støa was the co-supervisor and both him and Balasingham have been very helpful and supportive throughout the whole period. I would like to thank them for all their support and advice.

Oslo, July 9, 2007

Loc Tan Vo

Abstract

Mobility and freedom are two driving forces for research in wireless sensor networks. Recent advances in hardware development, offering low-cost, low-power sensor devices open up for an exciting research field where only the imagination can stop us from developing exiting and revolutionary applications.

In the medical field sensor networks can be used to remotely monitor physiological parameters such as heartbeat or blood pressure or patients, and report to the hospital when some parameters are altered. This is a huge research field, and much research will be done in the coming years to cope with recently unsolved issues.

This Master's thesis will present and describe a customized, tailor-made, and optimized cross-layer implementation of a communication protocol for medical purposes. The research platform is based on the IEEE802.15.4/ZigBee protocol that is widely popular for sensor networks applications. ZigBee based wireless sensor network finds interesting applications in medicine, and its flexibility, low cost, small hardware and low power consumption are promising features that could effectively serve medical applications.

Extended testing was necessary to address the implementation issues as well as finding optimization opportunities. The protocol has several exiting features, and experiments so far reveal very promising results.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abbreviations

ABP	Arterial Blood Pressure
ADMR	Adaptive Demand-Driven Multicast Routing
AM	Active Message
APS	Application Support Sub-Layer
BWSN	Biomedical Wireless Sensor Network
CBQ	CodeBlue Query Interface
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CVP	Central Veinous Pressure
DSSS	Direct Sequence Spread Spectrum
e.g.	(Exempli Gratia) For Example
ECG	Electro Cardio Gram
FCF	Frame Control Field
FFD	Full Function Device
FIFO	First In First Out
FTSP	Flooding Time Synchronization Protocol
GSM	Global System for Mobile Communications
GTS	Guaranteed Time Slot
IVC	The Interventional Centre

LDR	Link Delivery Ratio
LED	Light-Emitting Diode
LQI	Link Quality Indicator
LSB	Least Significant Bit
MAC	Medium Access Control
MIG	Message Interface Generator
MIG	Message Interface Generator
nesC	Network Embedded Systems C
NTP	Network Time Protocol
NWK	Network Layer
OQPSK	Offset Quadrature Phase-Shift Keying
OSI	Open System Interconnection
PDA	Personal Digital Assistant
PDR	Path Delivery Ratio
PHY	Physical
QoS	Quality of Service
RAM	Random Access Memory
RBS	Reference-Broadcast Synchronization
RFD	Reduced Function Device
RSSI	Received Signal Strength Indicator
SP	Sensornetwork Protocol
SQL	Structured Query Language
SYNC	Synchronization
TDMA	Time Division Multiple Access
TPSN	Timing-Sync Protocol for Sensor Networks
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Networks

ZDO	ZigBee Device Object

CHAPTER 1 Introduction

The world is going wireless. Today, mobile phones, Personal Digital Assistants (PDAs) and laptop computers are integrated parts of millions of people's everyday life. Wireless internet access, wireless keyboards and mice, remote controls etc. are more and more common. They are no longer considered sophisticated devices but are now a natural sight in homes and offices. Along with a wireless everyday life comes the advantage of mobility and freedom. These beneficial factors motivate for new and exciting wireless solutions and applications in untraditional contexts.

Recent advances in microsensor and radio technology will enable small but smart sensors to be deployed for a wide range of environmental monitoring applications. The low per-node cost will allow these wireless networks of sensors to be densely distributed [6]. These sensors can observe and react to changes in the physical phenomena of their surrounding environment and can be utilized in a vast number of applications. Only our imagination and ingenuity can put restrictions and limitations on the use of this technology. Amongst a diverse set of applications for sensor networks, some of the fields where this technology is most applicable are agriculture, environment, military, inventory monitoring, intrusion detection, motion tracking, machine malfunction, toys, medicine, and many other [3].

The wireless sensors are often organized in networks, including different network topologies and routing algorithms. The motivation is the ease of calibration of sensors and graceful degradation of services in case of breakdown of sensors [2]. Using some form of adaptive routing and self-organizing networks[3, 4, 6, 38, 43], new sensors are easily integrated in the network, and the cooperation can lead to greater understanding of physiological changes. Therefore, the network itself provides as a robust and effective early warning system.

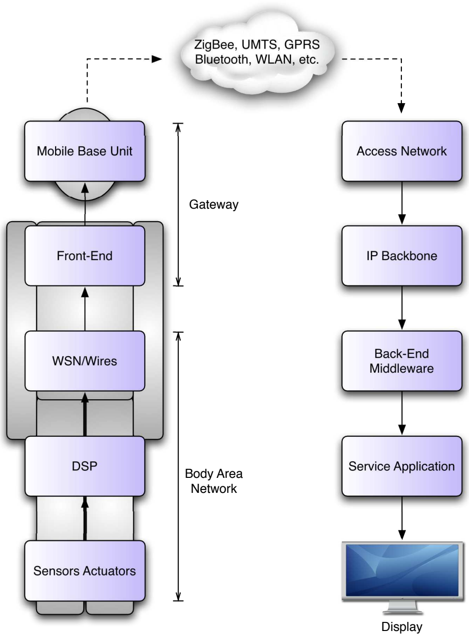


Figure 1.1: Overall System Design For Health Monitoring Systems

There are several profitable reasons for enhancing the medical field by the aid of wireless technology. Assuming a critically ill patient with several monitoring sensors attached to the body, the transport of this patient from one unit to another will be burdened by adverse events due to cable mess [2]. The avoidance of cable mess is evidently beneficial if multiple sensors are placed inside the body.

In hospital environments or clinics, equipping every patient with tiny, wearable wireless data acquiring sensors would allow doctors, nurses, and other caring assistants to continuously monitor the status of their patients. In emergency and disaster situations, the same approach would enable medical personnel to more effectively care for large numbers of casualties. By instantly monitoring the patient's respiratory, cardiac behavior etc., the hospital staff can effectively be distributed to the most critical patients.

Wireless sensors could enhance or replace existing wired telemetry systems for many specific clinical applications, such as physical rehabilitation or long-term ambulatory monitoring. Wireless monitoring over an extended distance can enhance freedom of movements in post operative periods, which in turn can lead to faster recovery. Patients who have recently been through a transplantation must be monitored closely to encounter possible discrepancy caused by the counteracting of the body after receiving a new organ. This requires the patients to regularly return to the hospital for post testings. A possible solution is to deploy sensors on the patients' body, which take samples and wirelessly send the data to a sink[1] node which in turn can send these information to the hospital. If everything is as expected, the patients don't need to return to the hospital. This will enhance the life quality of the patients, measured by increased mobility and freedom without reduction of safety. Hence, such a wireless home monitoring solution will lead to physiological and psychological synergy effects. Figure 1.1 shows the components that may be included in a health monitoring system.

Looking into the future, a complete remote health monitoring system could be economically very beneficial. Over the past years, an increase of chronicle diseases had been found in the population. Diabetes and heart diseases are lifestyle related illnesses that are growing rapidly, and currently, there are no indications that this trend will stop. Treatment and monitoring costs are high and this will introduce a great problem if every patient should be manually monitored by the hospital personal.

However, introducing wireless healthcare monitoring networks may also introduce new practical issues to be handled. Four fields of issues concern *Quality of Service (QoS), social acceptance, legal issues, and security issues* [2]. Wireless monitoring sensor networks may save lives, but they could also lead to fatal consequences should the networks break down. Ultimately high QoS is therefore required for networks in medical contexts. Social acceptance of the use of biomedical sensors includes health risks, economical, ethical, and organizational issues. We still don't know exactly how the body will react to the scenario of having sensors attached or even operated into the body for an extended period of time. Some people are also afraid that remote monitoring patients may violate the existing law of personal surveillance and the protection of patient sensitive data may be broken. This leads us to the legal and the security issues. Encryption of the data to be sent must therefore be implemented[2].

Despite the increased interests in wireless sensor networks, a significant gap still remains between existing sensor network designs and the requirements of medical monitoring. Many issues are to be considered in a sensor network design[1], and the different networks should be optimized for their usage area. To date, the primary design goal for sensor networks in general has particularly been energy efficiency, such as many of those presented in [24]. Most sensor networks are designed for stationary nodes that transmit data at relatively low data rates, with a focus on best-effort data collection

[1]A node that gathers information from other nodes and operates as an access point

[2][3, 20] give an extended discussion about security in IEEE 802.15.4/ZigBee standard.

at a central base station. However, as new applications of sensor networks emerge, other optimization criteria such as latency and compliance with real-time constraints, or reliable data delivery, such as medical applications, may gain importance. As opposed to usual data acquiring applications, medical monitoring requires relatively high data rates, reliable communication, and multiple receivers (e.g. multiple observers). Unlike many other applications applications, medical monitoring should make use of traditional in-network aggregation since combining data from multiple patients is not meaningful.

For medical sensor networks, it is sometime desirable to rather implement an own communication protocol than modifying existing protocols. As a contribution to the BWSN project, this master thesis therefore aims to tailor a communication protocol.

1.1 The Nordic BWSN Project

The Biomedical Wireless Sensor Network (BWSN) is a nordic collaboration including The Interventional Centre (IVC) at The National Hospital of Norway[3] and eight other partners. BWSN is intended to be used as part of a personalized wireless healthcare system. The project will focus on integration of existing sensors through efficient and secure wireless communication and development of business models for collaboration between the partners in the project. The final solution aims to involve examples with body sensors and implants communicating with the patient's mobile unit for personal control and external communication. Figure 1.2 demonstrates the BWSN system.

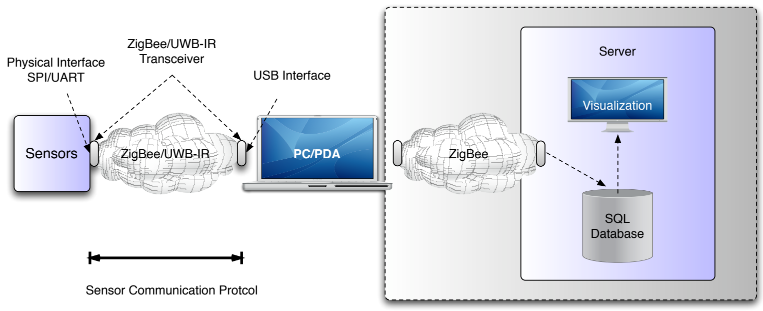

ZigBee/UWB-IR Transceiver
Physical Interface SPI/UART
USB Interface
Server
Visualization
Sensors
ZigBee/UWB-IR
PC/PDA
ZigBee
SQL Database
Sensor Communication Protcol

Figure 1.2: The BWSN Demonstration System

This master thesis' focus is to evolve and tailor a communication protocol for part of the BWSN project with the scope in the grey shaded area, enveloped

[3]Rikshospitalet - Radiumhospitalet

by the dotted line in figure 1.2. The protocol developed should be employed in self-configurated, self-organized, and scalable networks. In addition, the network should be able to adapt to continuously changes, e.g. new nodes starting up and joining the network. Hereafter, the protocol developed in this master thesis will be referred to as the "BWSN protocol".

The nordic project BWSN is extendedly described in [5].

1.2 Structure

In chapter two, a presentation of related work is made, including related biomedical systems. Further, in chapter three, the theoretical background for the implementation is presented including an introduction to cross-layer designing, followed by presentations of synchronization and roaming protocols suitable for the communication protocol in this thesis. Later, in chapter four, the theory regarding the system platform is presented. The IEEE 802.15.4/ZigBee standard is described, technical data regarding the testing equipments and scenario descriptions are provided. Chapter five explains the technical implementations made for the protocol, including software and experimental setups. Chapter six contains the results from testing and simulation, which are discussed in chapter seven. Finally, in chapter eight, concluding remarks and prospective work is presented.

CHAPTER 2

Related Medical Systems

Even though remote health caring systems are part of a new exciting and unexplored field, over the last few years some research has been done to realize these in practice. Some telemedical platforms such as SMART[41] and the MobiHealth Project[37] are research projects still going on. Since they both aim to use existing mobile technology to monitor patients, they are not very interesting for the BWSN protocol. [23], [29] and [33] present solutions more similar to what BWSN is searching, but still not spot on since their systems are also built for telemedical applications. So far, the closest related works are CodeBlue[26][40] and ZigMed[17].

2.1 CodeBlue

CodeBlue[26][40] is a project at Harvard University aiming to develop platforms for biomedical wireless sensor networks. The project provides protocols for device discovery and publish/subscribe multihop routing, as well as a simple query interface that is tailored for medical monitoring. Several medical sensors based on the popular Mica2[10], MicaZ[11], and Telos[34] mote designs are evolved, including pulse oximeter, Electro Cardio Gram (ECG), and motion-activity sensor. Even a miniaturized sensor mote is designed for medical use. CodeBlue includes a framework with protocols that perform device detections, multi-hop routing, and a simple query interface to provide easy data access for hospital staff. CodeBlue also integrates an RF-based localization system, called MoteTrack[25], to track the location of patients and caregivers. Overview of old and new publications can be found on the project's web page[45].

2.1.1 The CodeBlue Architecture

CodeBlue is implemented in TinyOs, further described in section 4.3, and provides protocols for integrating wireless medical sensors and end-user devices such as PDAs and laptops. The routing framework is based on a receiver initiated structure where the sensors are broadcasting and informing about which information they provide. The observers then ask for specific information and get them. This routing scheme is usually referred to as publish/subscribe routing and is useful in medical applications where data from different patients are overlapping to help the caregivers choose what kind of data they wish to acquire.

The Routing Layer

CodeBlue's routing protocol[7][40] is based on the Adaptive Demand-Driven Multicast Routing (ADMR)[22] protocol. The ADMR was chosen because it represents a fairly sophisticated and mature ad hoc multicast routing protocol, and was developed by an independent research group. It is simple and has been extensively studied in simulations. ADMR establishes multicast routes by establishing which nodes that are going to be forwarders on particular channels. These nodes will forward all messages they receive on their given channel, avoiding the same message to be sent multiple times from a "publisher" if there are several observers. This is the benefit of multicast routing. Nodes are elected and established as forwarders through a route discovery process that is initiated when a patient device requests to publish data.

The route discovery in ADMR works as follows: each node in CodeBlue maintains a node table indexed by the publisher node ID. Each node table entry contains the path cost from the publisher to the current node, as well as the previous hop in the best path from the publisher. Whenever an ADMR message is received, the new previous hop and path cost fields are updated accordingly. By letting the nodes periodically publishing which information they offer, the route discovery is updated and maintained. CodeBlue has an update interval of 15 seconds, which is long enough to let the network adapt to changes in the topology as a result of mobile nodes and other unpredictable factors.

When an observer wishes to acquire data from a certain channel, it sends a *unicast route reply* to the publishing node after calculating the best route from the node table. After receiving the *route reply*, all nodes on the way to the addressed node will configure themselves as forwarders for this channel. Due to asynchronous connections, some nodes will receive messages from nodes to whom they can't reply, which in turn leads to dropping of the *route reply* message. Therefore, the protocol employs only retransmission during the process of forwarding *route reply*.

For the routing costs, CodeBlue uses an estimator of the total Path Delivery Ratio (PDR) from the originating node. This is based on an empirical model that maps the CC2420[1] [21] radio's Link Quality Indicator (LQI) to an estimated Link Delivery Ratio (LDR). The total path loss can be calculated as the product of all LDRs for all links along the path from the originator to the current node. The PDR is included in the header of each ADMR message, and is updated incrementally for each hop. Hence, the path cost is 1-PDR which also is the *path loss ratio*. By providing this information, each node knows the best (lowest cost) path from each publisher to itself.

Discovery Protocol

In order for CodeBlue nodes to discover each other and determine the capabilities of each sensor device, a simple discovery protocol is layered on top of the ADMR framework. In CodeBlue, each node periodically publishes its own information, including ID and sensor type. ADMR supports that each channel can perform a simple flooding mechanism which broadcasts the information to all nodes in the network. Nodes that wish to know more about other nodes in the network, can subscribe to the broadcast channel providing the information they need. CodeBlue updates the metadata every 30 seconds.

CodeBlue Query Interface

The CodeBlue Query Interface (CBQ) layer allows receiving devices to establish communication pathways by specifying the sensors, data rates, and optional filter conditions that should be used for data transfer. The CBQ is generated by the observer, e.g. a PDA or a laptop, and instructs the nodes to publish data that meets the query conditions on a certain ADMR channel. The queries are fed into the network over broadcasting channel used in ADMR which see to that all nodes are receiving the message. Since the queries appear relatively seldom, they are spread in the network by flooding. The maintenance of dedicated routes would be a heavier load than a rare flood. The queries are broadcasted periodically until all receivers specified

[1]Texas Intruments' 2.4GHz IEEE 802.15.4 RF transceiver.

in them have given feedback. Each query has a unique ID which also contains the observers ID, preventing queries from different observers to create conflicts. By employing such an ID as a parameter, the observer can conveniently discard the query by a short message, should this be required.

2.1.2 CodeBlue Evaluation

The CodeBlue design was extendedly evaluated in [40] by using practical experiments. The main aims of the tests were to address the effects of scalability, latency and jitter, fairness, and mobility had on the protocol. So far, test results for Codeblue are promising, showing that CodeBlue and ADMR achieve good packet delivery ratios with modest data rates. However, radio bandwidth saturation is a serious problem with higher data rates, suggesting that this should be a primary focus for future work. More results from CodeBlue will be shown in chapter 6 as the results from tests of the BWSN protocol will be compared to CodeBlue, using CodeBlue as a benchmark.

2.2 ZigMed

ZigMed[18] is an application developed by Hansen and Støa as part of their master's thesis regarding a practical evaluation of IEEE 802.15.4/ZigBee Medical Sensor Networks. Using TinyOs and actual medical data rate, they performed experiments including single- and multihop using the MICAz[11] research platform. Data compression was also taken into consideration and expectedly the performance was better when less data was sent. They also employed a custom back-off algorithm to spread the motes' periodically sending time, reducing the collisions.

ZigMed did not implement a whole architecture such as CodeBlue, but the results and experiments are still very exciting. For comparative aids, the BWSN will also use the same data rates defined in ZigMed[18]. Some results from ZigMed will be presented in chapter 6 as comparisons to the BWSN protocol will be made.

CHAPTER 3 Theory

3.1 Cross-Layer Design

In traditional communication networks, the Open System Interconnection (OSI) layered architecture has been the most commonly used network model. Recent technological advances and the continuing quest for greater efficiency have led to an explosion in designs of wireless sensor networks. The layered architecture defines a stack of protocol layers in which each layer operates within its well-defined function and boundary. The great benefit of this approach is the allowing of changes to the underlying technology at each layer without imposing the need to change the overall system architecture. In addition to a successfully offer of transparency, such a layered architecture makes it easy to standardize. There are no doubts that this approach succeeds in the wireline networks, but it might not be totally suitable in the wireless networks domain.

Unlike the wireline networks, the wireless channel has several unique characteristics that need to be taken into consideration when designing wireless sensor networks. Summarized, designing for wireless networks poses more stringent requirements than wireline networks and the layered approach may lead to sub-optimal solutions and inefficient use of network resources. Therefore, new research has been conducted on whether a layerless approach may be more suitable for sensor networks, ending up in the new concept called cross-layer design[39, 28, 44, 12].

Cross-layer design and optimization is a new technique which can be used to design and improve the performance in wireless sensor networks. The central idea of cross-layer design is to optimize the control and exchange of information over two or more layers to achieve significant performance increase by exploiting the interactions between various protocol layers. Su and Lim[44] propose a cross-layer design and optimization framework, providing experimental results and analysis. The research done by Su and Lim shows promising results, and they strongly recommend cross-layer design as a new methodology for designing and optimizing the performance for future wireless networks because of the many possible benefits it could bring.

The idea behind cross-layer design fits the BWSN protocol perfectly well. When customizing and tailoring a protocol for a certain purpose, such as for the BWSN project, some desired functionalities and features are prioritized. Some functionalities offered in one certain layer in the OSI model may be more suitable to implement in a lower layer or vice versa in order to maximally optimize the protocol. By not strictly limiting functionalities to predefined layers, a greater flexibility in optimization procedures is offered. Therefore, in this thesis, I don't focus on a layer-like approach, but rather implement a general communication protocol.

3.2 Synchronization

Time synchronization is critical in sensor networks for diverse purposes. It is particularly important in a Wireless Sensor Networks (WSN) as its purpose usually is to correlate diverse measurements from a set of distributed sensors and synchronize clocks for shared channel communication protocols. Especially, precise timing is crucial for sensor data fusion, coordinations, and power-efficient duty cycling.

In a medical context, networks like BWSN require accurate time tagging on the data acquired, as their purpose is often to remotely monitor patients. Big variations in measured data due to drifting in global time is unacceptable as small differences in body signals may indicate irregular organ functionalities.

For years, Network Time Protocol (NTP)[30] has kept the most famous network, the Internet, in perfect synchrony. However, a new class of low-cost, large-scale and wireless networks requires a more precise synchronization than in traditional Internet applications. This is due to their close interactions with the physical world and their energy constraints. Research has been done to cope with the synchronization challenges that these networks evoke. Reference-Broadcast Synchronization (RBS)[13], Timing-Sync Protocol for Sensor Networks (TPSN)[15], and the Flooding Time Synchronization Protocol (FTSP)[27] are three protocols mentioned by Cox et al. [9] as adequate for time synchronization in ZigBee networks.

3.2.1 RBS

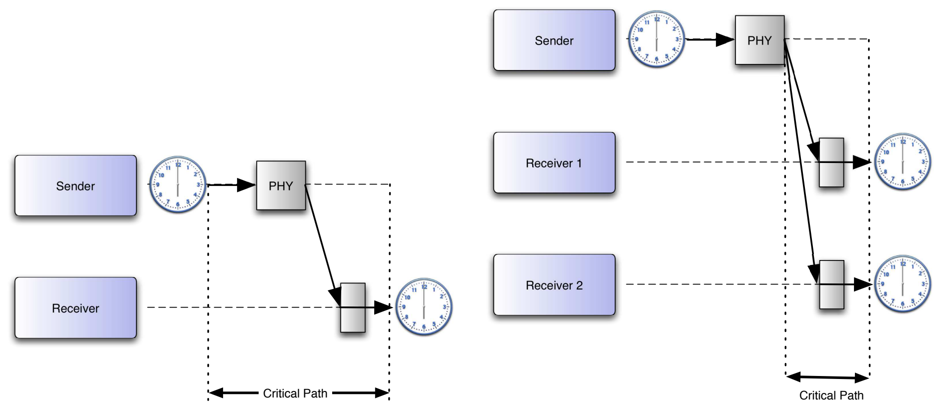
Most of the various network synchronization algorithms proposed over the years are using the same basic design. The main idea is to have a server periodically sending messages containing its current clock value to its clients. These protocols all have basic features in common: simple connectionless protocol; exchange of clock information among clients and one or more servers, methods for reducing the effects of nondeterminism in message delivery and processing, and an algorithm on the client for updating local clocks based on information received from a server. However, they can still differ quite much in detail.

Reference-Broadcast Synchronization (RBS) explores a form of time synchronization that differs from the traditional model. The fundamental feature of RBS is that it *synchronizes a set of receivers with one another*, as opposed to traditional protocols in which senders synchronize with receivers. In RBS nodes periodically send messages to their neighbors using the network's physical-layer broadcast. Recipients use the message's arrival time as a point of reference for comparing their clocks. The messages contain no explicit timestamp, nor is it important exactly when they are sent, at least not for RBS. By removing the sender's nondeterminism from the critical path[1] results in a dramatic improvement in synchronization over using NTP. RBS is applicable for a wide range of applications in both wired and wireless networks where a broadcast domain exists and higher-precision or lower-energy synchronization, than what NTP can typically provide, is required.

Non-determinism in the network is the main factor working against precise network time synchronization. Normal sources of errors are *Send Time*, *Access Time*, *Propagation Time*, and *Receive Time* [13]. Previous systems often cope with the sources of error by estimating and correcting them. RBS on the other hand does not estimate the error, it rather exploits the broadcast channel available in many physical-layer networks to remove as much of it as possible from the critical path. Elson et al. [13] observe that messages broadcasted at the physical layer will arrive at a set of receivers with very little variability in its delay. These observations are beneficial for the way RBS is working.

By making the recipients to synchronize with one another, we can remove the *Send Time* and *Access Time* from the critical path, Figure 3.1. In RBS, the critical path length is shortened to include only the time from the injection of

[1]The time it takes from the sender to receiver.



(a) Traditional Synchronization Protocols (b) Reference-Broadcast Synchronization (RBS)

Figure 3.1: Critical Path Analysis

the packet into the channel to the last clock read. This results in significantly better precision synchronization than algorithms that measure round-trip delay. The residual error is often a well-behaved distribution, which in turn can be limited by sending multiple reference broadcasts. Another advantage is that RBS allows nodes to construct local timescales. This is useful for sensor networks or other applications that need synchronized time but may not have an absolute time reference available.

Through simulations, Elson et al. show in [13] that RBS can synchronize clocks within 11 μs using Berkeley Motes with a relatively slow 19.200 baud radios. Tests on a 11Mbit/s 802.11 network synchronization was achieved of $6.29 \pm 6.45\mu$s. This is eighth times better than that of NTP[30] in a lightly loaded network. A heavily loaded network further degraded NTP's performance by a factor of 30, but had little effect on RBS. Including kernel timestamping, precision nearly reached the clock resolution of $1.85 \pm 1.28\mu$s.

Although showing impressive synchronization results, RBS still has some weaknesses and potential for improvement. Some important scaling issues have not yet been explored, such as automatic, dynamic election of the set of nodes to act as beacon senders. RBS also requires a network with a physical broadcast channel, which can turn out to be a significant limitation. It can not be used, for example, in networks that employ point-to-point links. RBS broadcast is always used as a relative time reference, never to communicate an absolute time value. This means that absolute time can't be achieved in the network unless one of the receivers is connected to an absolute time controller and then serves as a link between relative time and absolute time.

Additional message exchange is also necessary to communicate the local time-stamps between the nodes.

3.2.2 TPSN

Ganeriwal et al. present in [15] another protocol that aims at providing network-wide time synchronization in sensor networks. Timing-Sync Protocol for Sensor Networks (TPSN) is simple, scalable, and efficient. Moreover, TPSN is completely flexible and can easily be tuned to meet the desired levels of accuracy as well as algorithmic overhead. Ganeriwal et al. argue that for sensor networks, the typical approach of doing a handshake between a pair of nodes is better than synchronizing a set of receivers, as in the case of RBS. Time stamping the packets at the moment they are sent i.e. at MAC layer is claimed as a deciding factor for their statement. Unlike RBS, all nodes in the network synchronize their clocks to a reference node.

To facilitate deployment of MAC protocols such as Time Division Multiple Access (TDMA), there might be a need of maintaining a unique and global timescale throughout the network. In this case, TPSN creates a self-configuring system, suitable for sensor networks, where a hierarchical structure is established in the network. TPSN has its motivation from NTP that has been largely successful in the Internet. However, NTP is computationally intensive, which is not suitable for sensor networks where nodes are constrained by many factors, such as energy constraints. As distinct from NTP, TPSN can be seen as a practical, more accurate, and flexible extension of NTP to sensor networks.

Basically, TPSN works in two steps. First, a hierarchical structure is established in the network, and then a pair wise synchronization is performed along the edges of this structure to establish a global timescale throughout the network. In the first stage, also called the *level discovery phase*, every node in the network is assigned a level to create a hierarchical structure. A requirement is that a node belonging to level i can communicate with at least one node belonging to level $i - 1$. Only one node is assigned to level 0[2], which becomes the *root-node*. Once the hierarchical structure has been established, the root node initiates the second stage of the algorithm, which is called *synchronization phase*. In this phase, a node belonging to level i synchronize to a node belonging to level $i - 1$. Each node gets synchronized by exchanging two synchronization messages with its reference node on the other level. Eventually, every node is synchronized to the root node, and network-wide time synchronization is achieved.

[2]Which implicitly allows only one root node.

Ganeriwal et al.[15] perform theoretical analysis of errors in the communication schemes compared to RBS. TPSN has an added contribution from the uncertainty at the sender whereas RBS complete removes this as a source of error. From the analysis, TPSN still would give roughly a two times better performance for all the sources of error as compared to RBS. As a result, it has been shown that the classical approach of doing sender-receiver synchronization is a better approach than receiver-receiver synchronization in sensor networks.

For proving the above results, Ganeriwal et al. implemented both RBS and TPSN on Berkeley Motes and synchronization comparisons were made. The simulation results indicate 16.9μs and 29.13μs average error for TPSN and RBS respectively. While RBS had 93μs as worst case error, TPSN just had 44μs. In TPSN, the percentage of where time error was less than or equal to average error was higher (64%) than for the case of RBS (53%).

Simulations also demonstrated that the synchronization accuracy does not degrade significantly with the increase of numbers of nodes being deployed, making TPSN completely scalable. TPSN can be combined with the approach of post-facto synchronization to provide time synchronization among a subset of nodes. Post-facto synchronization is used to synchronize two nodes by extrapolating backwards to estimate the phase shift at a previous time. This is beneficial for multi-hop networks, and [15] shows that TPSN is suitable for these networks as well.

The drawback of TPSN is that it does not estimate the clock drift of nodes, which limits its accuracy. Moreover, it does not support dynamic topology changes and its performance was experimentally verified in a small multi-hop network only.

3.2.3 FTSP

Flooding Time Synchronization Protocol (FTSP)[27] is a robust time synchronization protocol tailored for applications with strict timing requirements and resource limited wireless platforms. This protocol was developed to provide network-wide time synchronization to a large network of wireless sensors. The FTSP was designed to accommodate network topology changes and to be robust despite the failure of individual nodes, which in turn are necessary considerations for a WSN.

The basic theory behind FTSP is to utilize concepts of MAC layer timestamping and skew compensation with linear regression. The first technique is used in TPSN[15] while the latter is employed in RBS[13]. Although these ideas are not completely new, their unique combination and its effective implementation yield better precision than existing approaches so far, according to [27]. As described in 3.2.1 and 3.2.2, both RBS and TPSN have improvement potentials.

Maroti et al. [27] gives a more detailed analysis on uncertainties in the radio message delivery, both deeper and wider than the analysis provided by [13] and [15]. A total of 10 error sources[3] were investigated, ranging from traditionally considered factors to newly introduced aspects. E.g. *interrupt handling time*, *encoding/decoding time*, and *byte alignment time* are mentioned and considered in neither RBS nor TPSN. [27] divides the sources of error into such many components in order to fully understand the uncertainty of the overlapping transmission and reception times, which both RBS and TPSN suffer from.

Using the error sources definitions in [27], Maroti et al. manage to express the weaknesses of RBS and TPSN distinctly and systematically. RBS is sensitive to the propagation, decoding, and interrupt handling time differences between the two receivers. The main source of error is the jitter in interrupt handling and decoding. TPSN on the other hand is sensitive to the encoding, decoding, and interrupt handling time differences between the sender and receiver. The common weaknesses for both RBS and TPSN are the jitter of interrupt handling and decoding time, which also are the two largest sources of uncertainty of MAC layer time stamping, Figure 3.2. By removing or reducing this uncertainty, FTSP may improve the performance significantly compared to both RBS and TPSN.

FTSP synchronizes the receiver to the time provided by the sender of the radio message. The broadcasted message contains the sender's timestamp which also is the global time. These messages are not dedicated synchronization messages; a timestamp is just appended to the regular broadcasted messages. The receivers get the corresponding local time from the local clock when receiving the message. Unlike RBS and TPSN, the time stamp of the sender must be embedded in the currently transmitted message. Therefore, the time stamping on the sender side must be performed before the bytes containing the time stamp are transmitted.

Wireless message transmission starts with the transmission of some preamble bytes, followed by Synchronization (SYNC) bytes before the actual data is appended. While transmitting the preamble bytes, the receiver radio synchronizes itself to the carrier frequency of the incoming signal. From the SYNC bytes the receiver can calculate the bit offset it needs to reassemble the message with the correct byte alignment. The timestamps are made after each byte boundary, after the SYNC bytes in transmit and receive.

[3]Send Time, Access Time, Transmission Time, Propagation Time, Reception Time, Receive Time, Interrupt Handling Time, Encoding Time, and Decoding Time.

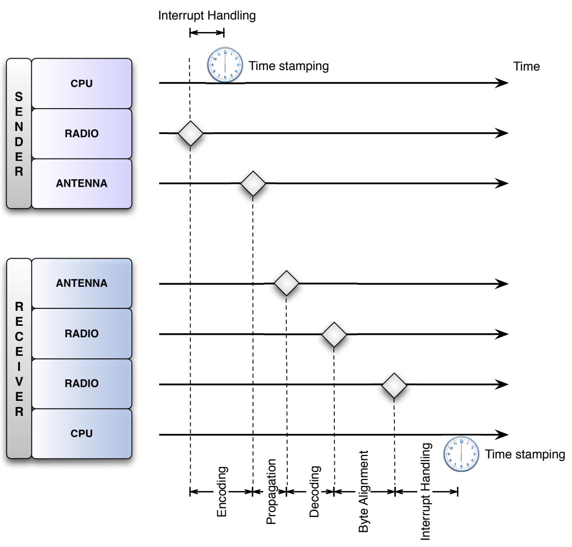


Figure 3.2: MAC Layer Time Stamping Uncertainty

The FTSP reduces the jitter of the interrupt handling and decoding times by performing time stamping like described earlier. Both the send and receiver side record multiple time stamps. These time stamps are then normalized by subtracting an appropriate integer multiple of the nominal byte transmission time, calculated from the transfer rate. The jitter of interrupt handling time can then be eliminated by taking the minimum of the normalized time stamps. Furthermore, the jitter of encoding and decoding time can be prevented by taking the average of the error corrected normalized time stamps. Besides, on the receiver side, the final averaged time stamp must be additionally corrected by the byte alignment time, which in turn can be computed from the transmission speed and the bit offset.

FTSP has several great features, such as tolerance towards topology changes and node and link failures. Periodic broadcasting of time synchronization messages handles the regular node and link failures, but FTSP also works for root node failure. Each node in the network remembers the most recent time when the root was active. If there have not been any root nodes for a certain time period, each node will time out and declare itself to be the root. If there are several root nodes, the node with the lowest local address will be elected. Nodes with higher local addresses trying to declare themselves as root node will then resign and become a normal node. To avoid the inconsistency in global time, nodes keep their old global time estimates and the new root sends its global time estimate instead of its local time as a new global time.

When a new node joins or leaves the existing network, the network topology changes. Assuming that the network remains connected at all times, a new node connected to it will wait for a certain time period, gathers data for the linear regression and determines the offset and skew of its own local clock from the global time. If the new node has the lowest address in the network, it will be elected as the new root, and continue sending correct global time that is close to the old global time.

Simulations[27] showed that for the single hop case using two nodes, FTSP had an average error of 1.48μs. Maroti et al. put up a 64-mote 7-hop network to simulate the behavior of FTSP. The average time synchronization error stayed below 11.7μs. Divided by the number of hops, the average error is as low as 1.7μs per hop. The maximum time synchronization error was below 38μs, observed only when the root was switched off. The simulations also showed no significant increase in time synchronization error during a node switch off or introduction of a new node.

Cox et al.[9] also implemented FTSP on the Telos platform[34] to investigate the performance of the protocol in ZigBee networks. In all cases, the slave node's error was never more than 61μs. Coupling the strengths of the FTSP and the master-slave configuration of ZigBee promises to be a profitable means of implementing time synchronization in WSN. The FTSP can be tailored to require even less from slave sensors while still providing the same degree of reliability and precision.

3.2.4 NetSync

NetSync is a synchronization protocol implemented in TinyOs, provided by Moteiv Corporation in their Boomerang[4] distribution. NetSync provides network-wide synchronization for Sensornetwork Protocol (SP)-enabled devices. It enables and maintains network synchronization for devices running the SP[36] link-layer abstraction. The SP layer is a newly introduced communication layer for sensor networks, placed between the network and link layers. It bridges the link and the network layers by providing link independent abstractions to build efficient network protocols. SP performs three main operations: data transmission, data reception and neighbor management.

NetSync uses several features offered by SP and is closely related to it. Neighbor management is used in NetSync to achieve network wide synchronization. The main idea behind NetSync, is to have one node as a time coordinator providing global time for other nodes in the network. When receiving a NetSync message containing the global time, the normal operating nodes will timestamp the message with their local time. An offset between

[4]Moteiv's business-ready open source for Wireless Sensor Networks

the local and global time will be calculated and stored. Later, when a global time is needed, the nodes can just translate the local time into global time by adding the offset already calculated. This offset will be updated periodically by the synchronization coordinator.

The network coordinator of the synchronized network is determined using a static procedure. It is the node that has a local address equals zero, in addition to being connected to a computer. NetSync uses a function in TinyOs called UartDetect to check whether the mote has established a connection to the computer. NetSync is implemented in TinyOs, and intends to be used within TinyOs. According to [35], the nodes using NetSync are synchronized to the order of about $100\mu s$ network wide.

Not much documentation is released on NetSync yet, and it is therefore not easy to understand the whole synchronization protocol. So far, for a better understanding of NetSync, it has been necessary to read the source code of the implementation.

3.2.5 Discussion

Comparing the different synchronization schemes presented above, FTSP performs significantly better than both RBS and TPSN besides being more robust and adaptive to the environments. It is not computationally complex, and [27] claims that FTSP utilizes less network resources than both RBS and TPSN. By combining the ideas from the both well-known previous protocols, FTSP appears by far as the best fitting time synchronization protocol for Wireless Sensor Networks (WSN).

Theoretically, NetSync has the worst performance when it comes to accuracy in the synchronization network wide. It still has the great advantage of being implemented and served as a component in TinyOs. This means that TinyOs applications can just use it without implementing an own synchronization protocol.

The choice of synchronization protocol will at last depend on a tradeoff between several factors, e.g. synchronization accuracy, complexity and protocol overhead. Different factors are dependent on the application that needs synchronization and which priorities it has. For BWSN, the highest priority is data throughput and robustness. Synchronization is needed to provide a global time, used to tag the measured medical data acquired. It is important that all sensors are operating with the same global time, providing accuracy in data interpreting. Assuming measurements of a patient's ECG, Arterial Blood Pressure (ABP), and Central Veinous Pressure (CVP) simultaneously, it is important that these data are timestamped at the same time. Shifted versions of originally correlated data is not desirable, and in the worst case, they can indicate illnesses. In addition, different biological sensors often have different sampling rates, which automatically will result in some mismatch in visualization timing. If the synchronization timing is wrong, this mismatch will be even bigger. If the sensors are not synchronized, the mismatch will grow bigger and bigger as the sampling continues. It is therefore desirable that the global time is updated frequently enough.

In BWSN, the fastest sensor to be used, is an ECG sensor, sampled at 1000Hz. This means that one sample is produced every millisecond. The synchronization requirement is therefore reduced to less than one millisecond, and all protocols presented above perform better than this requirement. Since RBS does not operate with a global time, it is not suitable for BWSN. FTSP uses flooding, which in turn leads to higher protocol overhead and seems to be more complex computationally than for instance NetSync. This means that the nodes has to do much processing locally, which may not be a good choice due to the energy conservation requirement. TPSN and NetSync are using the same basic idea, but NetSync has the advantage that it is already implemented and provided in TinyOs.

NetSync will therefore be chosen as synchronization protocol for BWSN.

3.3 Roaming

The BWSN project requires the communication protocol to support patient mobility. For hospital environment, it is natural that inter-room mobility will occur. Several base stations are needed to cover larger areas consisting of operation rooms, hallways and intensive care units. To offer completely mobility between these rooms, the motes on the patients must be switching from base stations to base stations. Including several base stations will cause the network to be cellular-like, and a roaming procedure is necessary.

So far, there are very little research made on roaming in traditional sensor networks. These networks don't need roaming in general, as the motes usually serve as static data collectors and very little mobility occurs. Traditional sensor networks often make use of multi-hop if the receiver is out of radio range to the transmitter. When using multi-hop, it is required that all motes are connected to the network through at least one other mote. This is not desirable in medical sensor networks, as multihop can lead to an incredible huge load of traffic to and from nodes close to a base station. As medical sensors are acquiring vital data, sensor nodes should not be serving as gateways and routing points. In addition, multi-hopping will affect the end-to-end latency by the number of interleaving nodes, increasing the total delay time. Communication should happen with low latency, so that vital changes in the patient's medical condition can be observed immediately; this is especially important during operation and other critical situations.

As opposed to multi-hopping, roaming require each sensor mote to be in communication range to a base station, and not another mote. This is not unreasonable, as mobility will introduce changes in topology, and one patient's motes may be out of range to other patients' motes, making multi-hopping impossible. In order to not loose any vital medical data, it is more feasible to stay connected to at least one base station.

Since not many roaming protocols are employed in sensor networks, the inspiration has to be taken from the mobile phone world, e.g. Global System for Mobile Communications (GSM) where the term "roaming" originates from. In BWSN, roaming means switching between base stations, and this is analogue to cell switching in GSM, also referred to as "handoff" or "handover".

3.3.1 Handover

In a cellular system, each cell has one base station and when a mobile moves into a different cell during an existing call, or when going from one cellular system into another, handover occurs. A globally defined signal level P_{min} is specified as the minimum usable signal at the base station. A slightly stronger signal level P_{HO} is used as a threshold at which a handover is made[19], figure 3.3.

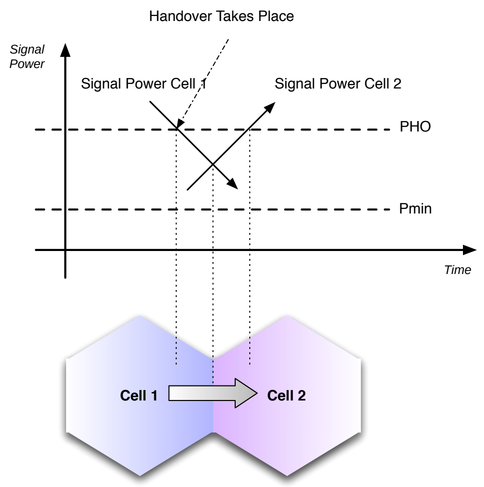


Figure 3.3: Handover

Each base station will constantly monitor the signal strengths of the received data to determine the relative location of each mobile user with respect to the base station. Based on this information, a decision will be made regarding handover or not. A calculated running averages should be used to avoid unwanted handover due to momentary fading.

In BWSN the Received Signal Strength Indicator (RSSI) offered by the radio chip can be used instead of signal power, and the same effect can be achieved. If the received signal strength indicated is less than a threshold compared to another base station, we can tell the mote to switch base station.

There are different ways of performing a handover and they are often divided into two categories, hard and soft handover.

Hard Handover

In hard handover, the old channel connection will be broken before the new allocated channel connection is set up, and for this reason, such handover is also known as "break before make", figure 3.4. Hard handovers are intended to be instantaneous in order to minimize the disruption to the call, but a call dropping obviously can occur.

Soft Handover

In soft handover, a "make before break" is done, which means that the new connection is established before the old connection is released, figure 3.4. The interval, during which the two connections are used in parallel, may be brief or substantial. A soft handoff may involve using connections to more than two cells, e.g. connections to three, four or more cells can be maintained at the same time. In such a soft handover state, the best connection can be chosen, or all connections can be combined to create a stronger link.

Discussion

Hard handover has the advantage of always having one connection established to one base station. This does not require the hardware to be able to keep up two connections simultaneously. A disadvantage is that if a handover fails, a connection will temporarily be disrupted, or even terminated abnormally. Therefore, when using hard handover, a procedure for re-establishing a connection has to be implemented, in the case connection was lost. If a connection cannot be re-established, vital data may get lost.

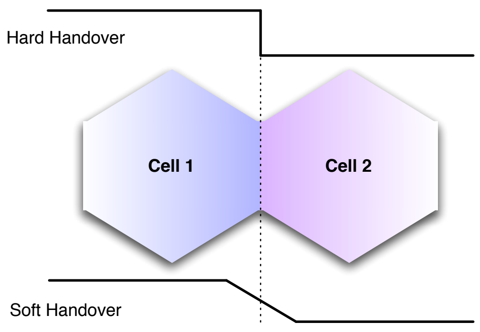


Figure 3.4: Hard and Soft Handover

One advantage of the soft handover is that the existing connection to a base station is broken only when a new reliable connection to another base station has been established. This significantly reduces the chances for connections to terminate abnormally due to failed handover. However, by far a bigger advantage comes from the mere fact that several connections are maintained simultaneously, and the communication can only fail if all the connections are interfered or faded out at the same time. If each connection uses its own channel and frequency, the interference in the different channels are unrelated, and therefore the probability of loosing total connection is very low. Thus, the reliability of the calls in soft handover is higher than for hard handover. The disadvantage is that each mote in the network has to be able to establish several connections as the same time. This require the motes to be more intelligent leading to more computational locally at each mote. If different channels are used at the same time, the motes has to be able to communicate on different frequencies at the same time. The motes to be used in the implementation later, section 4.2.1, only support one communication frequency at the time.

CHAPTER 4

System Platform

Through research[31, 17, 18, 46], the wireless standard, IEEE 802.15.4/ZigBee, has been identified as a promising candidate to be used in body area sensor networks. It is therefore chosen as the wireless standard to be used in the practical implementation of the BWSN protocol.

4.1 IEEE 802.15.4/ZigBee

IEEE 802.15.4/ZigBee define together a whole protocol stack for a new low-rate wireless network standard designed for automation and control network. The standard is aiming to be a low-cost and low-power solution for systems consisting of unsupervised groups of devices in houses, factories, and offices. It is expected to be used in applications for building automation, security systems, remote control, remote meter reading, and computer peripherals [32].

However, ZigBee has the potential to serve other application fields as well. The standard low-power solution and network organization abilities make it interesting for the use in biomedical sensor network. Everything used for biomedical purposes has to fulfill an extremely difficult requirement specification. Some of these are the reliability, launching time, network organization, and power consumption. A comparative study of WLAN, Bluetooth, and ZigBee for biomedical sensor networks was done in [17]. The results show that ZigBee has promising specifications when it comes to power consumption and launching time, which will be crucial for medical purposes.

ZigBee has significantly quicker launching time and lower power consumption, which will lead to a longer battery life-time, than both WLAN and Bluetooth.

4.1.1 ZigBee Specifications

The ZigBee standard uses the IEEE 802.15.4 standard as radio layer: MAC and Physical (PHY). Three radio bands (Figure 4.1) are defined:

- *Global:*
  ISM 2.4 GHz band with *16 channels* and a data rate of *250kb/s*
- *USA and Australia:*
  915 MHz band with *10 channels* and a data rate of *40kb/s*
- *Europe:*
  868 MHz band with *single channel* and a data rate of *20kb/s*

The defined channels are numbered 0 (868 MHz), 1 to 10 (915 MHz), and 11 to 16 (2450 MHz). Due to the protocol overhead, the actual data rates will be lower than the above mentioned [32]. The maximum defined length for an IEEE 802.15.4 packet is *127 bytes* including the header and the 16-bit checksum (CRC). The data payload is up to *104 bytes*.

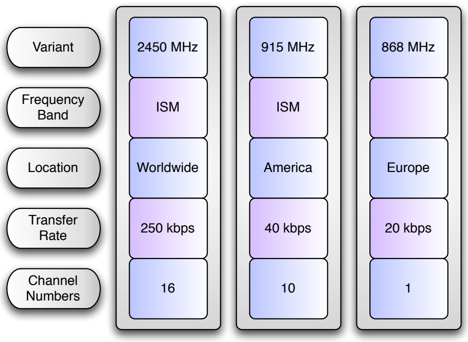


Figure 4.1: ZigBee Standard Baseband Variants

The IEEE 802.15.4[20] standards include optional acknowledgment mechanism to provide acknowledged transmissions if needed. This is done in the MAC layer only. The ZigBee stack, Figure 4.3, or the application has to take care that all data received by the MAC layer will be processed. The acknowledgment mechanism also handles retransmissions if acknowledgment is not received within a pre-defined time.

There are two types of devices defined in the ZigBee architecture. *Reduced Function Device (RFD)* is a device that has limited resources and does not allow some of the advanced functions included in the ZigBee standard, due to the fact that it is a low cost end device solution. Amongst the functions that a RFD can't offer, is network routing. *Full Function Device (FFD)* is a device that fully defines the ZigBee functionality and can become a network coordinator. The coordinator acts as the administrator and takes care of organization of the network. Typically, a coordinator holds a neighbor table of devices found in the neighborhood. This requires the coordinator to have more memory and processing power. The main issue is usually the memory.

4.1.2 ZigBee Network

Each ZigBee network can have a network coordinator if desirable. The coordinator starts the network, takes care of the structure and controls the procedure of assigning devices to and from the network. Every device that does not belong to a network has to go through a network association procedure. This includes that the device starts by sending an authentication request. The coordinator will answer this request within a predefined time. If a device intends to rejoin a network, it has to start the same procedure. An equivalent procedure is defined for the process of leaving a network. The device then issues a disassociation request [32].

Network Configurations

The IEEE 802.15.4 employs a long *64*-bit address and a short *16*-bit address [32]. Theoretically, the short address supports over *65 535*[1]. Nevertheless, this number is the theoretical number of the address space. How many devices each ZigBee network actually can handle depends on the multiple access scheme each network employs, and the degree of saturation when it comes to radio space. Each device also has a network identification (PAN ID) used to distinguish between overlaying networks. This PAN ID is a 16-bit number. To join a network, the device has to know the PAN ID of the network it wants to associate.

ZigBee also provides an optional super-frame structure with beacons for time synchronization. It also provides a Guaranteed Time Slot (GTS) mechanism for high priority communications. The time synchronization between devices in a beacon enabled network is performed by listening to the beacons transmitted by the coordinator. This enables the device to sleep for long periods, as the beacons can be set between 15 ms and approximately 4 minutes. By

[1]Since $2^{16} = 65536$, but address $0x00$ is reserved for the coordinator.

implementing this communication scheme, the ZigBee devices can perform significant power saving[32].

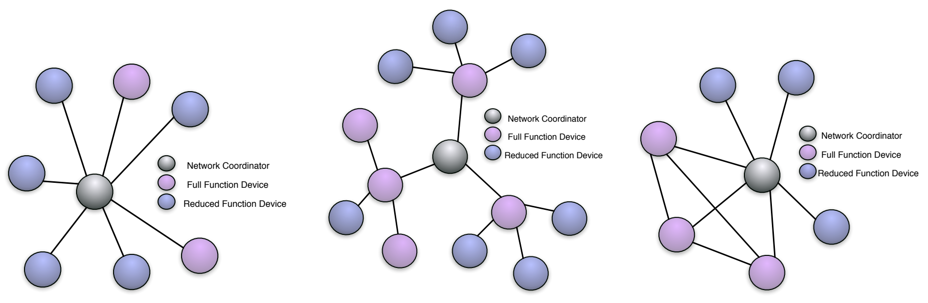
Network Topology

The ZigBee standard is in the first place intended for mesh networking. However, there are three main network topologies defined for the ZigBee standard.

The star topology, figure 4.2(a), is a network constructed as a single-hop topology with one coordinator in the center and the end devices. The devices in the star topology can only communicate via the network coordinator. This topology is necessary for networks containing RFDs, since they are not able to provide routing.

The tree topology, figure 4.2(b), is a multiple star topology with one central node that is the ZigBee network coordinator.

The mesh topology, figure 4.2(c), can also be called cluster-tree. Here, the FFDs in the network may communicate without the need of going through the coordinator. This network consists of star clusters, where only RFDs participate through a FFD as a router device, and of mesh links between the FFD devices. The star links are necessary for the RFDs in the mesh topology as they are able to communicate with a single FFD only.



(a) ZigBee Star Network (b) ZigBee Tree Network (c) ZigBee Mesh Network

Figure 4.2: ZigBee Network Topologies

At present, most of the ZigBee stacks are preliminary, and the support for multi-hop topologies is limited, but the base mesh functionality is usually supported. The preferred topologies are the mesh and star. Mesh topology enables flexible network configuration and provides redundancy in the available routes.

Data Transmission

Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)[2] is the default medium access method in the non-beacon enabled network. This means that any node may start the transmission at any time, as long as the channel is idle. In addition, Direct Sequence Spread Spectrum (DSSS) modulation method is used.

In the beacon enabled environment, the nodes are allowed to transmit in predefined time slots only. The coordinator sends a beacon frame (superframe) and nodes are expected to synchronize to this frame. The superframe includes time slots during which the nodes may compete using the CSMA/CA algorithm and may have special allocated slots for guaranteed access to the medium [32].

Power Consumption

The ZigBee standard is designed for applications that need to transmit small amounts of data while being battery powered so the architecture of the protocols and the hardware is optimized for low power consumption[3] of the end devices [32].

This is a great advantage when it comes to using ZigBee devices in biomedical environments. The battery life time requirement is essential in order to avoid the necessity of frequent battery changes. Using a ZigBee device with conservative power consumption is both adequate and desirable. The implementation of network topologies has also a great impact on the power consumption. By enabling beacons, the network can allow RFDs to go into sleep-mode and wake up on beacons from the network coordinator at predefined time slots. An extended discussion on power reservation implementations is provided in [3].

4.1.3 ZigBee Protocol Stack

In addition to the Physical (PHY) and the Medium Access Control (MAC) defined by the IEEE 802.15.4, the ZigBee Protocol Stack consists of the Network Layer (NWK), the Application Support Sub-Layer (APS), and the ZigBee Device Object (ZDO). Refer to Figure 4.3 as the protocol stack is better visualized.

[2]Please refer to [20] as the CSMA/CA algorithm is described

[3]Texas Instrument's ZigBee-ready chip, CC2420, has a power consumption corresponding to 18.8 mA in receive mode, and 17.4 mA transmit mode (at 0 dBm) [21]

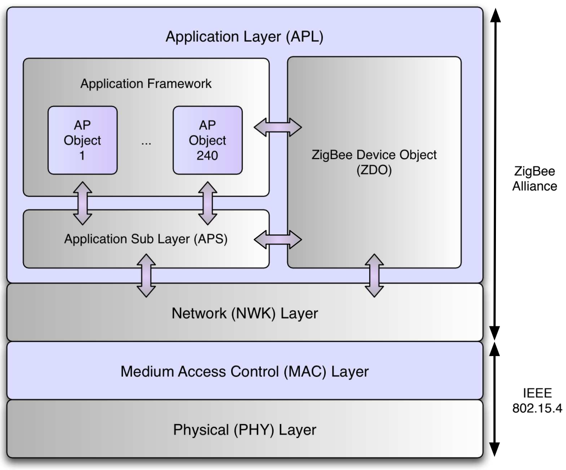


Figure 4.3: Full ZigBee Protocol Stack

Physical (PHY) Layer

This is the lowest layer in the protocol stack and is defined by the IEEE 802.15.4[20] standard. The Physical (PHY) Layer deals with encoding and decoding of bits that are sent over the communication channel. In addition, it provides the MAC Layer some information about the channel as clear channel assessment, link quality indication, and the received signal strength indication [32].

Offset Quadrature Phase-Shift Keying (OQPSK) is used in the PHY Layer in the 2.4GHz band. A Direct Sequence Spread Spectrum (DSSS) modulator is used to modulate the raw data. In the 2.4GHz band, the chipping rate is 2 million chips per second. By combining these chips, we get a raw data rate of 250 kbps. The modulation is simpler in the 868 and 915 MHz band, and the data bits simply alter the instantaneous phase of the carrier[20].

Medium Access Control Layer

Medium Access Control (MAC) Layer defined by the IEEE 802.15.4[20] standard, controls access to the shared radio channel. The layer generates and recognizes the addresses in addition to verifying the frame check sequences. The general packet format is shown in Figure 4.4.

The MAC layer also handles the transmissions of frame in the non-beacon mode. When the beacon-mode is enabled, there is an optional superframe structure, Figure 4.5, that can be used to guarantee access to the channel if required. This superframe is initiated by the beacon received by coordinator,

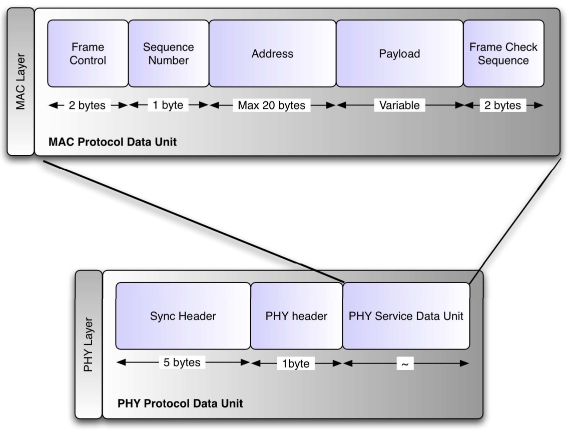


Figure 4.4: General Mac Packet Format

and followed by 16 equal time slots. The first nine slots can be used by any device, and the following seven slots are Guaranteed Time Slot (GTS), which can be reserved and be allocated to individual devices by request.

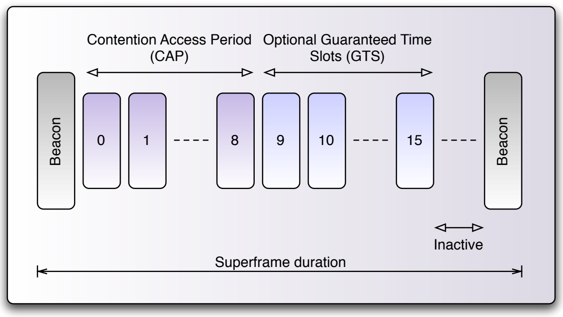

Contention Access Period (CAP)
Optional Guaranteed Time Slots (GTS)
Beacon
0
1
8
9
10
15
Beacon
Inactive
Superframe duration

Figure 4.5: ZigBee Superframe

One significant detail to notice is that beacons are not supported by the mesh topology. The consequences are that beacons then can not be used for synchronization purposes [32].

Network Layer

As the name reveals, this layer handles the network level of the communication. It handles routing and security functions for relayed messages in addition to keeping up the network structure. This is where the network keeps and maintains information about the nodes within the network [32]. The network properties and parameters are defined in the application as the configuration of the ZigBee stack. These properties may include topology, security, etc. [3, 32].

Application Layer

This layer consists of the user defined code of the individual custom application [32]. The application layer also defined whether the ZigBee device is FFD or RFDs, and it also deals with the security[3] functions and the behavior of the device in the network when certain system events occur. The Application Support Sub-Layer (APS) contributes as a low-level application layer. It has the ability of discovering and binding neighboring devices in addition to the responsibilities of forwarding messages among devices that are not able to communicate directly.

4.2 Moteiv Research Platform

The practical implementation in this thesis is based on hardware products from Moteiv Corporation, a leading provider of wireless sensor network solutions. They provide innovative hardware platforms, robust open-source software, and whole-solution development services. All software for motes developed in this thesis are programmed and intended to run on Moteiv's Tmote Sky, functionally equivalent to Moteiv's Telos Revision B, which is maybe more familiar. Another product to be used is Moteiv's Tmote Connect, further described in section 4.2.2, serving as base station connection points in experiments described in section 5.2.

4.2.1 Tmote Sky

Tmote Sky[8] is a reliable IEEE 802.15.4 research platform. It is referred to as the next-generation mote platform for low power, high data-rate sensor network applications. Tmote Sky employ the largest on-chip Random Access Memory (RAM) size of any mote, has an integrated on-board antenna providing up to 125 meter range. The radio included on the mote is the well known CC2420 IEEE 802.15.4 radio chip from Texas Instruments. As opposed to for example MICAz[4], Tmote Sky provides an easy-to-use Universal Serial Bus (USB) protocol for programming, debugging and data collection.

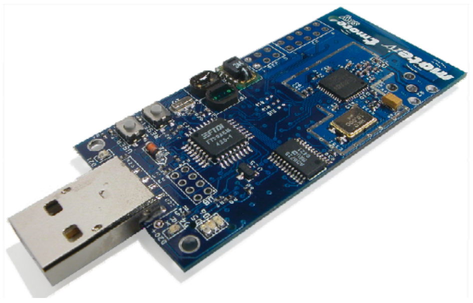


Figure 4.6: Tmote Sky from Moteiv

Tmote Sky supports TinyOS out-of-the-box, and TinyOS applications can easily be compiled and run on these motes. Together with the motes, they also provide Boomerang, Moteiv's business-ready open source for WSNs. It is fully compatible with existing TinyOS 1.x applications, and including key features from TinyOS 2.x. As mentioned above, Tmote Sky is functionally

[4]IEEE 802.15.4 research platform from Crossbow Technology Inc.

equivalent to Moteiv's Telos Revision B, allowing all Telos Revision B code to be run without modifications on the Tmote Sky module.

Tmote Sky uses the CC2420 Radio chip from Texas Instruments, a wideband radio with OQPSK modulation together with DSSS at 250kbps in the 2.4 GHz band. It offers encryption and authentication, packet handling support, auto acknowledgments, and address decoding. It also offers a feature for "reserved frame mode", which is a key feature in the roaming protocol to be implemented and described in chapter 5.

Signal Quality Measurements

For a later development of the BWSN protocol or any other WSNs, it is important to understand and know the performances of the radio chip, e.g. mote coverage range and signal strength. The CC2420 radio chip provides information about Received Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI) to give an indication on the correlation between received signal strength and distance.

Received Signal Strength Indicator (RSSI) The CC2420[21] has a built-in RSSI given a digital value read from the eight bit, signed twos complement register *RSSI.RSSI-VAL* on the chip. The RSSI value is always averaged over eight symbol period (128μs), in accordance with [20]. In TinyOS, the RSSI of each packet is provided and can easily be accessed for each packet received. The RSSI value from the CC2420 radio chip is very linear in the dBm-scale, and has a dynamic range of about 100dBm, from 0 to -100dBm.

Link Quality Indicator (LQI) The LQI measurement is a characterization of the strength and/or the quality of the 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.

The LQI value is required by [20] to be limited to the range 0 through 255, with at least eight unique values. The appropriate scaling of the LQI has to be done in software, and the calculation of LQI can be done directly by using the RSSI value, a correlation value given by CC2420 or a combination of both. Using the RSSI value directly to calculate the RSSI has the disadvantage that e.g. a narrowband interferer inside the channel bandwidth will increase the LQI, giving a false positive. The true link quality is actually reduced. If the correlation value is directly used, a value of 110 indicates a maximum quality frame, while a value of 50 is typically the lowest quality frames detectable by the radio chip. The correlation value is an average made over the eight first symbols in the packet.

The LQI value in combination with the RSSI are valuable to determine communication links with high probability of success. In BWSN, if a roaming protocol is chosen that involves link decisions when it comes to switching between base stations, LQI and RSSI values are truly helpful.

4.2.2 Tmote Connect

In BWSN, the base stations are also Tmote Sky modules. For large are testing with several base stations, described in more details in chapter 5, Tmote Connect are used as connection points for these Tmote Sky modules, allowing them to be connected and monitored through Ethernet infrastructure.

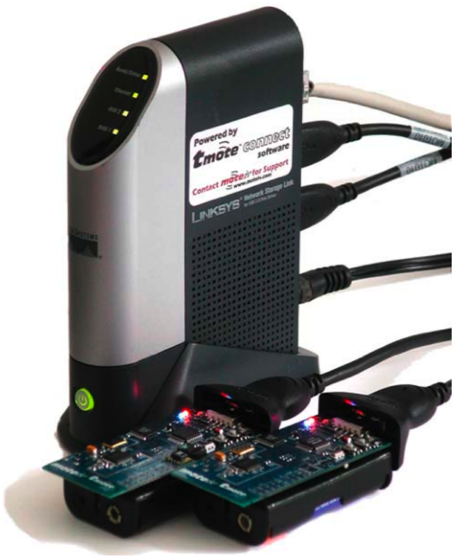


Figure 4.7: Tmote Connect from Moteiv

Tmote Connect acts as a wireless gateway applicance, connecting Tmote wireless sensor modules, e.g. Tmote Sky, to a wired local area network. Each Tmote wireless module connected to a Tmote Connect can be remotely administered through a web-based graphical user interface. In addition, it fully integrates with TinyOS development system and tools, allowing no additional configuration to be made.

4.3 TinyOS

TinyOS[14] is a small, energy-efficient operating system developed by a consortium led by the University of California, Berkeley in co-operation with Intel Research. It is an open source component-based operating system and platform targeting WSNs. It is designed to be able to incorporate rapid innovation and implementation while minimizing code sizes, as memory constraints often cause problems in sensor networks. TinyOS' component library includes network protocols, distributed services, sensor drivers, and data acquisition tools. The operating system deploys an event driven execution model which allows flexibility scheduling, which is often necessary in wireless communication as they are facing many unpredictable factors.

4.3.1 Network Embedded Systems C (nesC)

The TinyOS operating system, libraries, and applications are all written in nesC[16], a dialect of the C programming language optimized for the memory limitations of sensor networks. nesC is a new structured component-based language that is primarily intended for embedded systems. It has a C-like syntax, but supports the TinyOS concurrency model, as well as mechanism for structuring, naming and linking together software components to create network embedded systems.

nesC highlights the use of interfaces, and each component can use and provide several interfaces. The provided interfaces are intended to represent the functionality that the component provides to its user, the used interfaces represent the functionality the component use to perform its job. By implementing interfaces, components are statically linked to each other. This increases runtime efficiency and better the code generation.

CHAPTER 5

Implementation

This chapter gives insight to the implementations made in BWSN. Choices made during the implementations will be presented and discussed.

5.1 System Design

When making a customized protocol design for a certain purpose, the protocol should be optimized according to the priorities pre-defined for the system. As mentioned in [1], there are many issues involved when designing a protocol for wireless sensor networks, and usually a tradeoff between many non-congruent factors, such as energy-efficiency, network reliability, robustness and scalability has to be considered. BWSN prioritize network reliability and network robustness while optimizing for data throughput, data delivery and mote mobility. Some design decisions are made based on theoretical analysis and research done by others.

BWSN does not support multi-hopping, and there are no intercommunication between nodes directly. Sensors are acquiring medical data, and their task is to send it to the base stations. Previous simulations[18] shows that multi-hopping can significantly reduce the data deliver ratio. For BWSN, data deliver ratio is extremely important, so multi-hopping will not be implemented.

5.1.1 BWSN Overview

In BWSN, the communication links can be described by figure 5.1. All communication between the motes and the server is done through the base station. The base station is simply a Tmote Sky module connected to a Tmote Connect. Radio packets received from the base station will either be visualized by the server, or stored in a SQL database, or both.

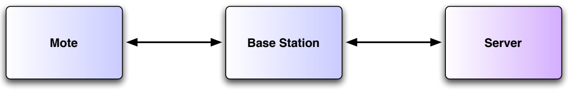


Figure 5.1: Overall System Setup

Software on the Motes

The application installed on the Tmote Sky module consists of several components from the TinyOS library. Some of these components are original, and some are modified or added. Basically, the software for the motes in the network are based on existing TinyOS programs, such as Oscilloscope or OscilloscopeTmoteSky. They all have the main function of sending messages to the base station.

For the different experiment setups described in section 5.2, several versions of the program developed exist, since small modifications were made to cover all the test cases. All versions of the programs developed are distributed in appendix B.

Software on the Base Station

The software running on the base station is a modified version of TinyOS application, TOSBase. TOSBase acts as a simple bridge between the serial and radio links. When a server wants to send out a message, TOSBase will forward this message to the radio link and send it to motes with the same group ID. Equivalently, it listens to the radio link and filters out messages that do not contain the same group ID as its own. TOSBase includes queues in both directions, with guarantee that once a message enters a queue, it will eventually exit on the other interface. For all tests in done for BWSN the maximum queue size of 256 is used. This is done by including the line CFLAGS+=-DTOSH_MAX_TASKS_LOG2=8 in the TOSBase Makefile.

Software on the Server

The server is an application implemented in Java based on TinyOS Oscilloscope application. For communication with the base station, TinyOS offers an application called SerialForwarder to forward the packets from TOSBase attached to the Tmote Connect and the server running on a computer. TinyOS also includes a tool called Message Interface Generator (MIG) for nesC. MIG generates code that processes TinyOS messages and creates Java objects based on the original message defined by the mote application. This allows the Java application to receive and send TinyOS messages by wrapping and unwrapping them as Java objects.

The standard Oscilloscope application visualizes the oscilloscope messages received by TOSBase. An own class, BWSN.java, is implemented to achieve additional functionalities required to control the network. In addition to visualizing the data, the server now distributes network addresses and controlling which which motes exist on the network. If a mote requests to start up with an address already taken on the network, the server can distribute a new address to it. The server is the brain in the BWSN network.

TinyOS Message Structure

The Tmote Sky IEEE 802.15.4 compatible message structure is defined by the structure TOS_Msg, which can be found in the AM.h file in TinyOS CC2420Radio component.

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. When a message is received, the received event associated with that handler ID is signaled. This can be seen as a soft message filter.

Each AM type used in the BWSN application are defined as message structures that are embodied in the TOS_Msg data payload at sending. Sandvand from Memscap[1] defines in [42] the differentAM types to be used in BWSN.

5.1.2 Start Up Procedure

When a node starts up, it is important that it will be operating as quick as possible. Hence, the start up procedure should be efficient with low latency, but at the same time be secure and robust. In BWSN, each mote starting up

[1]Sensor producer, also a partner in the BWSN project

```
typedef struct TOS_Msg{
  /* The following fields are transmitted/received on the radio. */
  uint8_t       length;         // Length Of The Packet
  uint8_t       fcfhi;          // Frame Control High Byte
  uint8_t       fcflo;          // Frame Control Low Byte
  uint8_t       dsn;            // Data Sequence Number
  uint16_t      destpan;        // Pan Destination Address
  uint16_t      addr;           // Destination Address
  uint8_t       type;           // AM Type
  uint8_t       group;          // Group ID
  int8_t        data[TOSH_DATA_LENGTH]; //Data Payload

  /* 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 Of Received Packet
  uint8_t       lqi;            // LQI Of Received Packet
  bool          crc;            // Calculated CRC
  bool          ack;            // ACK Request
  uint32_t      time;           // Time, units in 32khz
} __attribute((packed)) TOS_Msg;
```

Figure 5.2: TinyOS Message Structure

in the network will send a BootIndicationMsg to the server to indicate start up, announcing information about itself. The mote will wait for an answer from the server, providing the information needed. The start up procedure can be clearest described by the flowchart for both the mote and server in figure 5.3.

The messages involved in the start up procedure are BootIndicatinMsg and ConfigureMsg. Their structures can be seen in figure 5.4.

When a server received a BootIndicationMsg, it will interpret the information, especially sourceMoteID and BWSNchannel. The BWSNchannel is a number telling which channel this sensor is using and has nothing to do with the radio channel. BWSNChannels are distributed to all sensor producers, ranging from 0-499. Each sensor is assigned one BWSNchannel, and the channel number uniquely identifies the sensor.

The server will check whether the address requesting to join the network already exists or not. Since the start up address for a mote is pre-compiled, mistakes can occur, and several motes could have the same address, i.e. sourceMoteID. If a mote with ID A is already active on the network, and another mote with ID A, called A', the server will tell A' to change its address to the new address stored in newSourceMoteID in the corresponding ConfigureMsg. The corresponding ConfigureMsg will be sent to A, and both A and A' will receive this message. To solve the potential problem that both motes

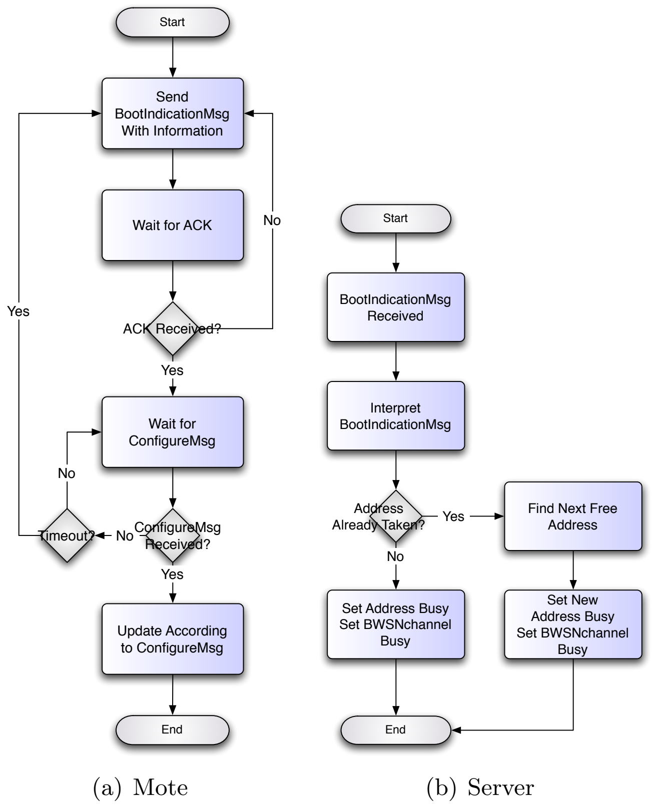

Start
Send BootIndicationMsg With Information
Wait for ACK
No
ACK Received?
Yes
Yes
Wait for ConfigureMsg
No
Timeout?
No
ConfigureMsg Received?
Yes
Update According to ConfigureMsg
End
(a) Mote
Start
BootIndicationMsg Received
Interpret BootIndicationMsg
Address Already Taken?
Yes
Find Next Free Address
No
Set Address Busy Set BWSNchannel Busy
Set New Address Busy Set BWSNchannel Busy
End
(b) Server

Figure 5.3: BWSN Start Up Flowchart

```
typedef struct BootIndicationMsg{
  uint16_t sourceMoteID;                    // Original Source Address
  uint16_t BWSNchannel;                     // BWSN channel
  uint16_t status;                          // Currently Unused
  uint32_t timestamp;                       // Timestamp
  uint16_t SamplerateTimesHundred;          // Sample Rate
  uint8_t majorVersion;                     // Currently Unused
  uint8_t minorVersion;                     // Currently Unused
} BootIndicationMsg_t;
```

```
typedef struct ConfigureMsg{
  uint32_t timestamp;                       // Timestamp
  uint16_t SamplerateTimesHundred;          // Sample Rate
  uint8_t RadioChannel;                     // Radio Channel
  uint16_t newSourceMoteID;                 // New Source Address
} ConfigureMsg_t;
```

Figure 5.4: Message Structures In Start Up Procedure

A will change their addresses, they are controlled by a states. While waiting for a ConfigureMsg, the mote is in a STATE_DEF_START_UP_PENDING, while a normally operating mote is in STATE_DEF_OPERATING. If a mote is not in STATE_DEF_START_UP_PENDING, it will automatically discard the ConfigureMsg received, and it will therefore not change its address.

To differ between motes that received a new address and motes starting up normally, the address distributed by the server is starting from 500 and above.

5.1.3 Synchronization

As discussed in section 3.2.5, NetSync is chosen as the synchronization protocol used in BWSN. It has the timing accuracy required for BWSN, and it has the advantage of already being implemented in TinyOS. NetSync is offered as a synchronization component to be included in normal applications, and its functionalities are provided through the *GlobalTime* interface.

```
interface GlobalTime<precision_tag>{
  async command uint32_t get();
  async command global_time_t getBoth();
  async command uint32_t convertToGlobal(uint32_t local);
  async command uint32_t convertToLocal(uint32_t global);
  async command bool isValid();
}
```

Figure 5.5: GlobalTime Interface

NetSync requires the mote with address 0 to be the synchronization coordinator, and this mote has to be connected to a computer or a Tmote Connect. Since NetSync uses the Sensornetwork Protocol (SP)[36][2] it can easily detect neighbors on the network, and communicate with them. It also uses its own message structure, and the communication and exchanges of NetSync messages are hidden for the user.

NetSync can be easily included in an application by typing make ⟨platform⟩ lowpower when the custom program is compiled. When a synchronization coordinator exists on the network, the *GlobalTime* interface is used to get the global time of the network. The global time is received and ready to be accessed when *isValid()* returns true. *GlobalTime* also provides methods for converting global time to local time and vice versa, based on an offset that is automatically calculated.

[2]Briefly described in section 3.2.4.

5.1.4 Roaming

The roaming protocol implemented for BWSN is the heart of the communication protocol, and the BWSN's performance is highly dependent on the roaming strategy chosen. BWSN requires a high and reliable throughput in addition to mobility support. As discussed in section 3.3.1, BWSN will make use of several base stations, and a handover strategy is necessary.

Soft handover is chosen for BWSN. For medical sensors, vital data is acquired and sent to the base station, and the communication protocol can not afford to loose connection during base station switching. For soft handover, all motes are communicating with one or several base stations simultaneously, and thus the chances for a broken connection link is minimized. By maintaining several links to the base stations, the chances for successful data delivery is increased.

Assume two neighboring base stations with overlap in radio range, denoted as *BS1* and *BS2*. If the traffic load at *BS1* is much higher than *BS2*, a mote *M* within *BS1*'s communication area can send the data to both *BS1* and *BS2* simultaneously. Combining the data received from both base stations will enhance the packet delivery ratio. This of course requires that *BS2* is in the range of *M*.

To optimize the data throughput, the handover time has to be kept low, and the roaming protocol should not introduce much overhead. One disadvantage of soft handover is usually the complexity required to keep connection with several base stations. It is therefore important to keep the implementation as simple as possible to not require the motes to perform extensive handover algorithms.

Figure 5.6 shows the roaming scheme implemented in BWSN. All base stations will accept all messages sent by the motes, and duplicate entries will be filtered out[3].

The roaming protocol implemented in BWSN is very robust, stabile, and flexible . It successfully removes the handover time, introduces no roaming overhead, and the motes do not perform any handover computations. In fact, the motes never know that they have changed base station. Since switching from one base station to another is totally costless, mobility is welcomed by the roaming protocol implemented.

The basic idea is to use a form of "selective broadcast". All motes will use a broadcast scheme that allows only base stations to receive these messages,

[3]At present time, this filter is not yet implemented. All packets received are stored in the Structured Query Language (SQL) database, and duplicate entries are post processed by queries made to the database.

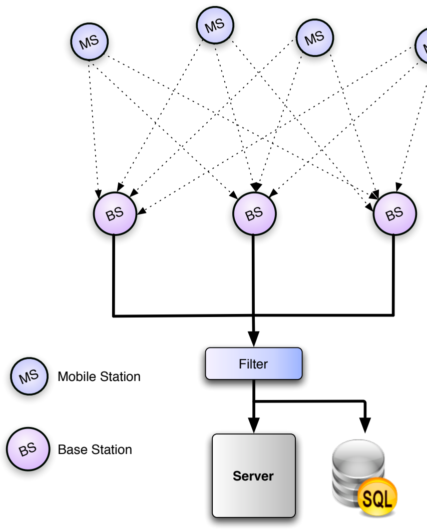


Figure 5.6: Roaming Scheme

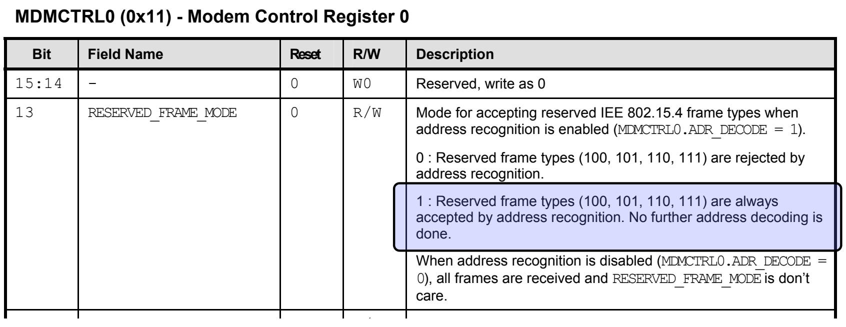
and no other motes on the network will listen to them. The ”selective broadcast” is realized by using a hidden feature in the CC2420[21] radio chip, referred to as ha *RESERVED_FRAME_MODE*. TinyOS does not provide this feature, and modifications of the communication module implemented in TinyOS are necessary.

Radio Chip Settings

The CC2420 radio chip has many registers that can be modified. The values stored in these registers determine the chip’s behavior and configure its settings. *MDMCTRL0* is the first modem control register for the radio chip. Features as address decoding, automatic Cyclic Redundancy Check (CRC), and automatic acknowledgment are all controlled by this register. It also controls the *RESERVED_FRAME_MODE* feature, which is utilized in BWSN. Figure 5.7 shows the part of the *MDMCTRL0* that regards *RESERVED_FRAME_MODE*.

From figure 5.7, it is clear that if bit number 13 in the *MDMCTRL0* is set to 1, the radio will accept packets with frame types 100, 101, 110, and 111 without performing address decoding. All base stations in BWSN enable *RESERVED_FRAME_MODE* by writing a new value to the *MDMCTRL0* register in TinyOS setting bit 13 to 1, and all other motes set this bit 0. If bit 13 equals 0, all packets with frame type 100 to 111 will be discarded even though the destination address is correct.

In addition, all motes that are not base stations have to change the frame

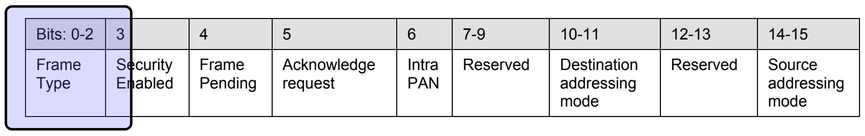


MDMCTRL0 (0x11) - Modem Control Register 0

Bit	Field Name	Reset	R/W	Description
15:14	-	0	W0	Reserved, write as 0
13	RESERVED_FRAME_MODE	0	R/W	Mode for accepting reserved IEE 802.15.4 frame types when address recognition is enabled (MDMCTRL0.ADR_DECODE = 1). 0 : Reserved frame types (100, 101, 110, 111) are rejected by address recognition. 1 : Reserved frame types (100, 101, 110, 111) are always accepted by address recognition. No further address decoding is done. When address recognition is disabled (MDMCTRL0.ADR_DECODE = 0), all frames are received and RESERVED_FRAME_MODE is don't care.

Figure 5.7: Part Of The MDMCTRL0 Register[21]

type of all messages to a binary value between 100 and 111. The frame type is defined by the three Least Significant Bits (LSBs) in the Frame Control Field (FCF), figure 5.8. In TinyOS, the FCF is defined by fcfhi and fcflo in the TOS_Msg structure shown in figure 5.2.



Bits: 0-2	3	4	5	6	7-9	10-11	12-13	14-15
Frame Type	Security Enabled	Frame Pending	Acknowledge request	Intra PAN	Reserved	Destination addressing mode	Reserved	Source addressing mode

Figure 5.8: Frame Control Field[21]

Changing register value and setting the fcfhi value correctly is not straightforward. The source code provided in appendix B is demonstrating how this is done.

TinyOS Modifications

Changes were made to *CC2420RadioM* module in TinyOS in order to make the roaming protocol work. *CC2420RadioM* is a huge module and implements the MAC layer in TinyOS. It does not really offer the features already existing in the chip except from acknowledgment, and this feature is also very statically implemented. To know whether the acknowledge request bit in FCF is set or not, instead of just performing check on that single bit, TinyOS try to match the whole FCF field with some predefined values. When these values were defined, no consideration were taken to the remaining fields of FCF. This means that different *frame types* are not supported.

Therefore, the modifications were made to *CC2420RadioM* to make it more flexible, and all modifications made are backward compatible. The new version of *CC2420RadioM* module will work perfectly fine with other existing

TinyOS applications running on a Tmote Sky module. It is also provided in appendix B.

5.1.5 General Optimizations

A few modifications were made to optimize TinyOS to perform even better than it is at present time. After digging deep into how TinyOS works, and the data sheet of the radio chip, it was clear that TinyOS could be further optimized. The radio chip is offering several deft features that are not utilized. Too much of the functionalities are done by software, and the optimization consists of moving these to the hardware.

TinyOS perform address checking in software. Each packets that has the same group address and passes the CRC are interpreted in the software. If the destination address is correct, it passes the packet further to the application. The radio chip offers hardware address decoding, and by using this, the chip checks the destination address of the packet in hardware. If the packet is not intended to be received, the radio chip flushes the receive FIFO buffer where the packet is stored, without generating any interrupts to the software. The software can then concentrate about other tasks without being interrupted. In addition, the delay of packet processing is reduced.

When a packet is received with the *acknowledge request* bit set, the software will interpret the packet and then send a strobe commando to the radio chip. The radio chip will generate an acknowledge packet and send it in return. Once again, it is the software that is performing the main part. By turning on the automatic acknowledgement in the radio chip, an acknowledge packet will be generated directly by the hardware if it sees that the *acknowledge request* is set. The whole mechanism will work faster and ensure that the mote that requires acknowledgment does not time out.

Both hardware address decoding and automatic acknowledgement can be turned on by modifying the *MDMCTRL0*[21] register in the radio chip.

5.2 Experimental Setup

Physical Attributes

In all tests for BWSN, *radio channel 26* is used. In IEEE 802.15.4/ZigBee, the center frequency of a given channel is calculated as:

$$F_c = 2405 + 5(k - 11), \quad \text{where } k \,\epsilon\, [11, 26] \tag{5.1}$$

where k is the channel number. This channel is outside of Wireless Local Area Network (WLAN) interference, and previous work[18][46] shows that the packet loss ratio is slightly lower in a channel with no WLAN interference. The same work shows that the packet loss ratio is lower for larger packets than for small packets. Therefore, the maximum size of *127 bytes* are used for all PHY packets in BWSN. The transmit power Tx is set to *0 dBm*, which is the maximum.

Medical Data Properties

In experiments, each mote will be regarded as one medical sensor. The three types of sensors are Electro Cardio Gram (ECG), Arterial Blood Pressure (ABP), and CVP. Their properties are described in table 5.1.

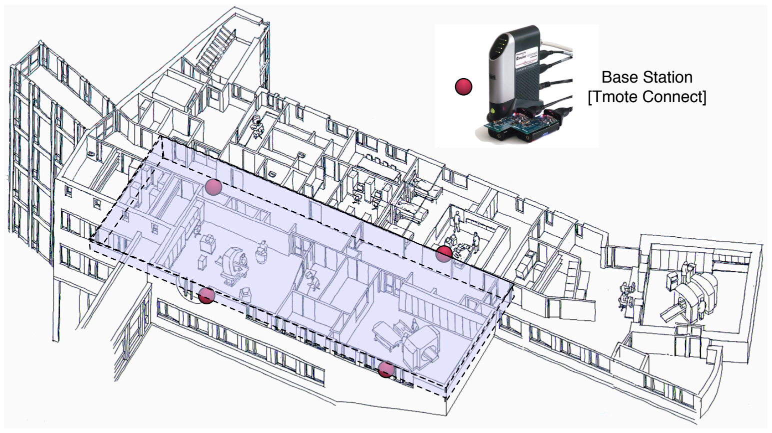
Data Type	**Sample Frequency**	**bit/sample**	**Data Rate**
ECG	200 Hz	16 bit	3.2 kbps
ABP	100 Hz	16 bit	1.6 kbps
CVP	100 Hz	16 bit	1.6 kbps

Table 5.1: Medical Data Properties

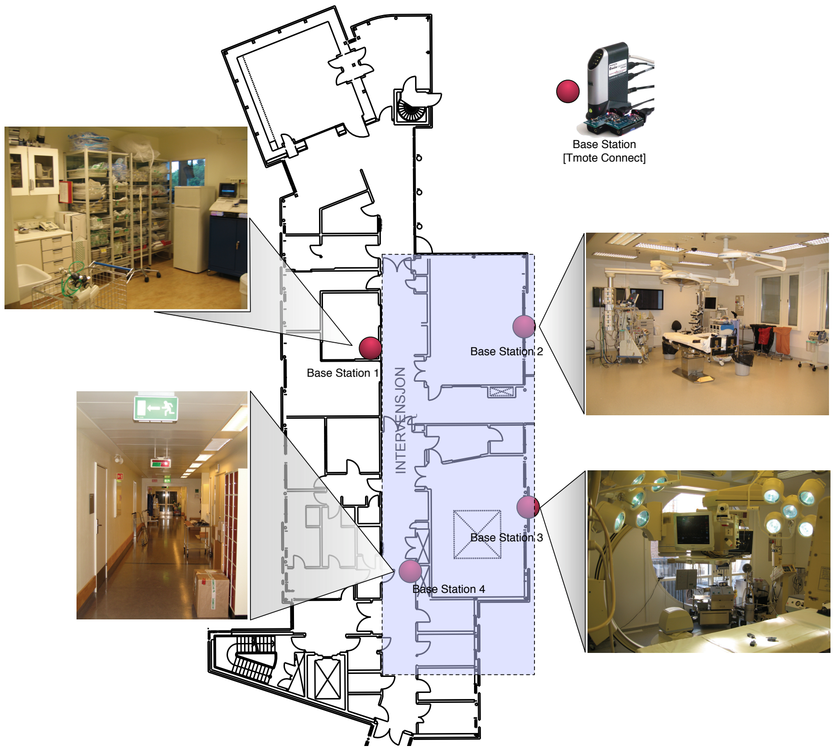
Test Environment and Base Station Placement

Except from the experiment one and two, section 5.2.1 and 5.2.2 respectively, all experiments were carried out on the $3^r d$ floor at the The Interventional Centre (IVC) at the National Hospital of Norway[4] to secure true hospital environment. Four base stations were used, and their placement at the $3^r d$ floor is shown in figure 5.9. The light blue shaded area is about $240m^2$ and all experiments were performed within that area.

[4]Rikshospitalet - Radiumhospitalet



(a) 3D Setup



(b) 2D Setup

Figure 5.9: Base Station Setup

5.2.1 Experiment 1: Start Up Procedure

This experiment aims to test the start up procedure implemented in BWSN. A quick start up is important, and it is desirable to se how much time our protocol spends on starting up. This was done by timestamping the mote during start up. Four timestamps were collected for each test, and the test was run five times to investigate the drift of the mote. The four timestamps were taken at following occasions:

1. *Timestamp 1*: Taken right before BootIndicationMsg was sent.
2. *Timestamp 2*: Taken right after BootIndicationMsg was sent.
3. *Timestamp 3*: Taken right after ConfigureMsg was received.
4. *Timestamp 4*: Taken when the start up procedure has completed.

5.2.2 Experiment 2: Base Station Bandwidth

For designing a complete system as BWSN, it is important to know approximately how much traffic load each base station can handle before it starts to drop packets. Knowing the bandwidth of the base station will be useful when planning how many base stations are needed to cover a certain amount of patients.

```
typedef struct OscopeMsg{
  uint16_t sourceMoteID;          // Original Source Address
  uint16_t lastSampleNumber;      // Last Sample Number
  uint16_t channel;               // Oscilloscope Channel
  uint16_t data[55];              // Data Payload
} OscopeMsg_t;
```

Figure 5.10: OscopeMsg

The test setup for this experiment is very simple, involving one base station and one mote. The mote is placed approximately 1.5m away from the base station, and at the same height. It is programmed to send a fix number of identical packets, but the sending frequency is increasing for each run to increase the data rate. The message structure of the sent packets is shown in figure 5.10. It contains 110 bytes of raw data, and the total overhead is 16 bytes including the overhead in TOS_Msg, figure 5.2. For each run, 11000 packets are sent.

5.2.3 Experiment 3: Base Station Coverage Range

Knowing the radio coverage range of the hardware is important. Since the roaming protocol in BWSN requires all motes to at least be in the communication range with one base station, this experiment should show approximately how many base stations are needed to cover a certain area. In addition, the test should show how obstacles and loss of line of sight will affect the packet reception ratio.

The test includes all four base stations, and the roaming protocol developed is used. The area shaded light blue in figure 5.9 is divided into 75 different coordinates. 50 packets are sent from each coordinate and received by one or several base stations. Packet reception ratio, RSSI and LQI will be calculated for each coordinate.

```
typedef struct OscopeMsg{
  uint16_t sourceMoteID;         // Original Source Address
  uint16_t lastSampleNumber;     // Last Sample Number
  uint16_t channel;              // Oscilloscope Channel
  uint8_t rssi;                  // Free Space for RSSI
  uint8_t lqi;                   // Free Space for LQI
  uint16_t data[53];             // Data Payload
} OscopeMsg_t;
```

Figure 5.11: OscopeMsg

The message structure of the sent packets is slightly different from what used in experiment 1. When TOSBase receives a message, it stores the RSSI and LQI values before storing these packets in the database. Two fields, rssi and lqi are added to provide free space for storing the RSSI and LQI values, figure 5.11. The data rate used was 3.2kbps *excluding overhead*, equivalent to an ECG sensor. Including overhead, the data rate was 3.875kbps.

5.2.4 Experiment 4: Medical Scenarios

Three medical scenarios are chosen to test the BWSN's performance in real-life and real environment. Differently from the previously described experiment, this test includes at least three sensors at a time. In all scenarios chosen, each patient is simulated by using three motes, sending ECG, ABP, and CVP data. The message structure is the same as in figure 5.11.

In this test, all motes will be waiting for a message telling them to start sending data. This message is injected into the network by a mote that does not take part in the experiment. It just sends out a message, and will never receive any of the messages from the other motes since it is not a base station. Assuming the motes receive the start up message at time t_0, and starts to send data after a time t_1. If all motes have the same t_1, the radio channel will get very saturated by packets being sent almost simultaneously if there are many motes. This will reduce the packet reception ratio, giving a false indication on how the system perform. Usually, sensors are started manually, and the time t_1 will automatically be different amongst the motes. Therefore, a preemptive backoff algorithm to spread their starting time is necessary, i.e. give them different t_1 values. Different strategies can be used, and one of them is to use a random timer to give each mote a random start up time, like [17] did. However, when using a random timer, the probability of two or several motes starting up closely to each other is increasing as more motes are involved.

To ensure that all motes are dynamically spread out in start up time, BWSN uses the following equation to assign start up time for the motes:

$$t_1 = TOS\_LOCAL\_ADDRESS \ \cdot \ \sigma \tag{5.2}$$

σ is a spreading factor. *TOS_LOCAL_ADDRESS* is the address each mote has. Since each mote has an unique address, their startup time will also be unique. In this experiment, the spreading factor σ is set to 25ms.

This experiment also tests which impact number of retries has on the packet reception ratio if no acknowledgment is received. First, each scenario will be tested without requiring acknowledgment and no resending of lost packets. Then, acknowledgment is required and the motes will try to resend lost messages 3, 6, and 9 times. For each scenario and each number of retries, three separate runs are made. Each run will last for at least 2 minutes. For all cases, all four base stations are used.

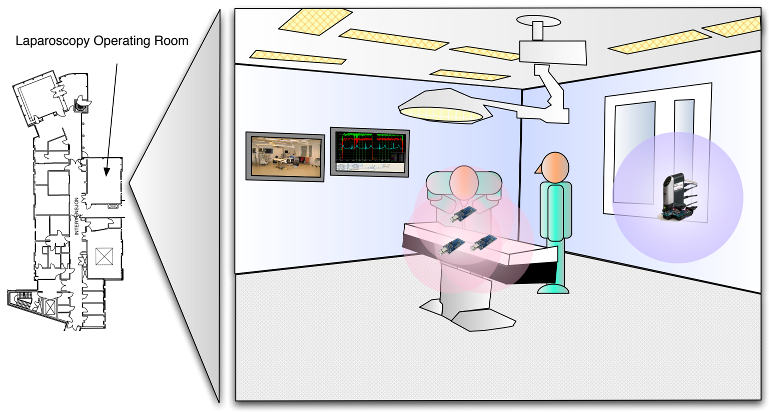
Operating Room

This scenario simulates a patient wearing sensors during an surgery. The laparoscopy operating room will be used for this test, figure 5.12(a), and it is approximately $30m^2$ big. Three motes are placed on a bed about one meter above the ground, and they have addresses from 5 to 7. Addresses from 1 to 4 are already taken by the base stations.

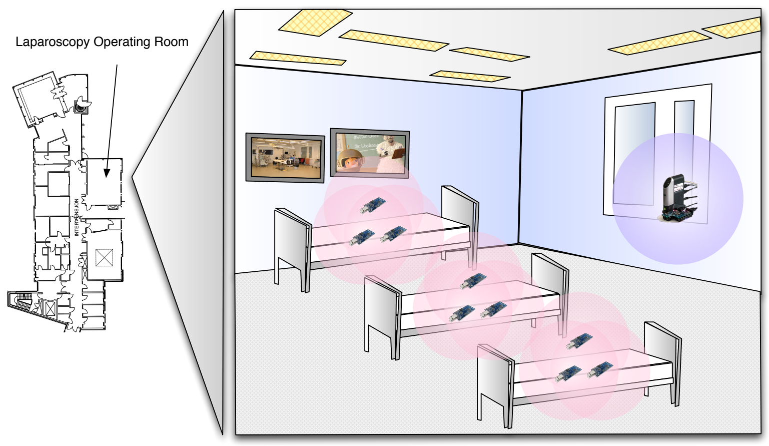
Intensive Care Unit

This scenario simulates several patients in an intensive care unit, figure 5.12(b). The same laparoscopy operating room is used for this test, and the placement of base stations is as described in figure 5.9. Now, nine motes are used to simulate three patients, with addresses from 5 to 13. Each group of motes consist of ECG, ABP, and CVP sensors.

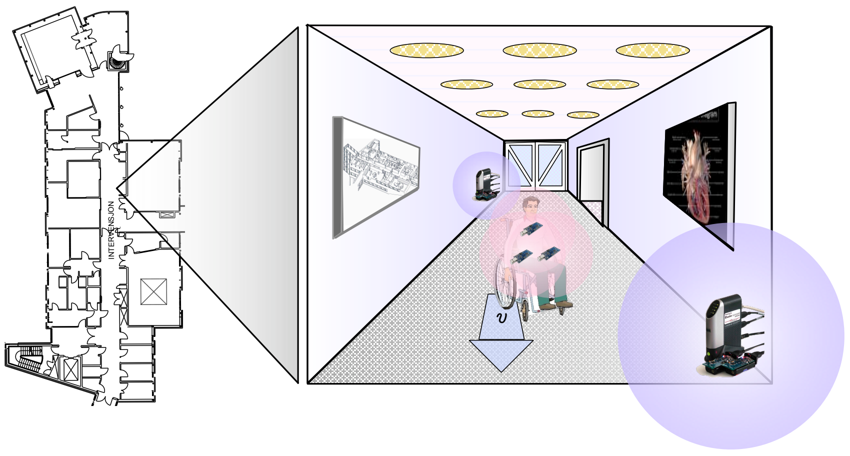
The purpose of this experiment is to test the protocol's scalability and the case of a large number of motes in a smaller area.



(a) Operating Room



(b) Intensive Care Unit



(c) Patient Mobility

Figure 5.12: Medical Scenarios

Corridor Patient Transport

This scenario simulates a patient movement in the corridor, figure 5.12(c). The experiment will test the protocol's ability to support movement and patient mobility. Three motes are used to simulate one patient, addressed from 5 to 7. These motes are moving with approximately 1.2 meter per second. They start at the end of the corridor near base station 1 and move towards base station 4. If the direction from base station 1 to base station 4 is defined as "down", then during one run, the motes are moving "down", "up", and "down" again.

CHAPTER 6 Results

In this chapter, results obtained from the different experiments in chapter 5. Matlab was used to perform SQL queries and process the data gathered from the database.

6.1 Synchronization

NetSync was chosen as synchronization protocol in BWSN due to the fact that it was already implemented in TinyOS and could easily be included in any applications. NetSync is completely hidden for the application that includes it, and the synchronization is performed automatically. It turned out that NetSync was too hidden and the synchronization happened too automatically.

Including NetSync in BWSN introduced several problems. The application did not behave normally anymore. In BWSN, packets are being sent with fixed interval to obtain a fixed data rate. The whole mechanism was controlled by a timer, and by regulating this timer, the data rate was regulated. When NetSync was included, this timer was somehow overruled by NetSync. Regardless of the timer value set in the application, packets were sent with the same constant rate, which was quite slow.

Almost no documentation on NetSync exists, and it was difficult to understand what actually happened. By extended searching on the Internet, it turned out that several people have experienced the same problems with NetSync earlier. NetSync is intended to be used in low-power mode only, and it will automatically reduce the duty cycle on the mote. The desired duty cycle value and be appended in the make-command which includes NetSync. Unfortunately, NetSync will never operate on a duty cycle higher than 50% of maximum.

In addition, it turns out that NetSync *always* includes NetWake, which is a global network wakeup protocol. By including NetWake, communication can only be done within fixed predefined time slots, and that's why the BWSN protocol could no longer define the data rate itself. Regardless of how often it tries to send a packet, the packet will be queued until the time slots occur.

Due to the lack of time, it was not possible to implement an own synchronization protocol. Attempts were made to use NetSync without NetWake, but a warning was stated about attempts to use NetSync separately:

> Always use NetSyncC by compiling with make ⟨platform⟩ lowpower. Directly wiring to NetSyncC or to NetWakeC can adversely alter the order of system initialization and produce unpredictable results.

Some attempts were made to rewrite NetSync, but due to the time limitation, no working solutions exist.

6.2 Roaming

The roaming protocol implemented works really well. Even though no experiment was designed to test the roaming protocol, it was extendedly tested by experiments 3 and 4, section 5.2.3 and 5.2.4 respectively. The handover procedure between base stations is really seamless and can be characterized as very soft. The motes do not recognize that they have changed base station, and the handover is totally transparent. No roaming overhead was introduced, and no further processing is done by the motes when they were connected to new base stations.

All base stations are accepting packets of frame types 100, 101, 110, or 111 regardless of the address decoding. Testing showed that if a packet with frame type 000, 001, 010, or 011 was sent to a single base station, this base station will still accept the packet if the address decoding succeeds. Since the frame type is dynamically changed in the software, the roaming protocol also supports uni-link communication. The motes can choose whether they want to send to all base stations or to choose one single base station. At time present, BWSN does not make use of this advantage, may at least the opportunity exists.

The optimizations made in TinyOS together with the roaming protocol implemented, enhance the communication protocol. Previously, if mote A is sending a packet to mote B, and mote C is in the range of A, C will load the packet and check the destination address before discarding it. With the optimizations including moving functionalities to hardware, and the use of *RESERVED_FRAME_TYPES*, now C will discard the packet already at hardware, without interrupting the software. Neighboring motes that are not communicating will no longer disturb each other at the same degree as previously.

6.3 Result Experiment 1: Start Up Procedure

For measuring the time a node uses to start up before being operative, four timestamps were made in the start up procedure. They are made at:

1. *First timestamp* t_0: Made right before BootIndicationMsg was sent.
2. *Second timestamp* t_1: Made right after BootIndicationMsg was sent.
3. *Third timestamp* t_2: Made right after the ConfigureMsg was received.
4. *Fourth timestamp* t_3: Made when the mote was ready to operate normally.

Run No.	t_0	t_1	t_2	t_3
1	0 ms	10.56 ms	64.76 ms	64.82 ms
2	0 ms	10.56 ms	70.16 ms	70.19 ms
3	0 ms	10.56 ms	64.42 ms	64.48 ms
4	0 ms	10.56 ms	63.05 ms	63.11 ms
5	0 ms	10.56 ms	75.13 ms	75.20 ms
Average	0 ms	10.56 ms	67.50 ms	67.56 ms

Table 6.1: Mote Start Up

This was done five times, and the results are shown in table 6.1. It took on average 67.56 ms from the BootIndicationMsg was successfully queued for sending until the mote finished to process the ConfigureMsg and ready to operate normally. From the different runs, the best result obtained was 63.11 ms, and the worst case was 75.20 ms. In all runs, the node start up procedures were always successful. None of the BootIndicationMsg, Acknowledgment, or ConfigureMsg were lost.

An interesting observation is that t_1 is constant for all runs. Since t_1 does not include any propagation time and waiting time for the server, this means that the time it takes for a mote to perform a certain amount of the code is constant.

6.4 Result Experiment 2: Base Station Bandwidth

In experiment two, the base station bandwidth was tested.

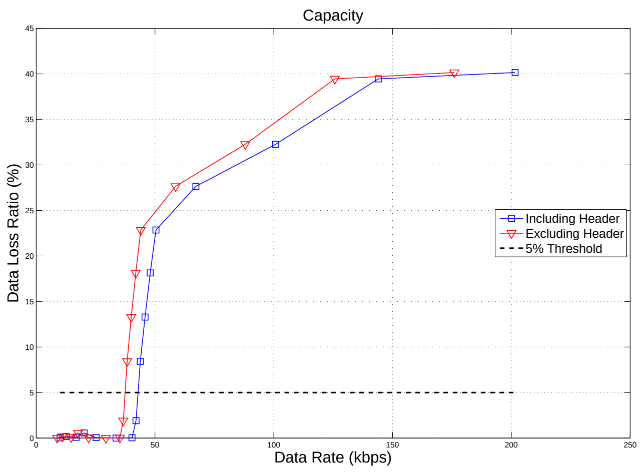


Figure 6.1: Base Station Bandwidth

The test result shows that for data rates less than approximately 40kbps, the packet loss ratio is insignificant, figure 6.1. When data rate increases beyond 40kbps, the packet loss ratio grows exponentially, before it flattens out around 130kbps and 140kbps. The TOSBase was programmed to blink the blue Light-Emitting Diode (LED) every time a message could not be queued for transmission over the UART link to the server, and had to be dropped. At data rate above 40kbps, the blue LED was blinking rapidly and spontaneously, indicating that packets were received at a rate higher than TOSBase could queue them for sending.

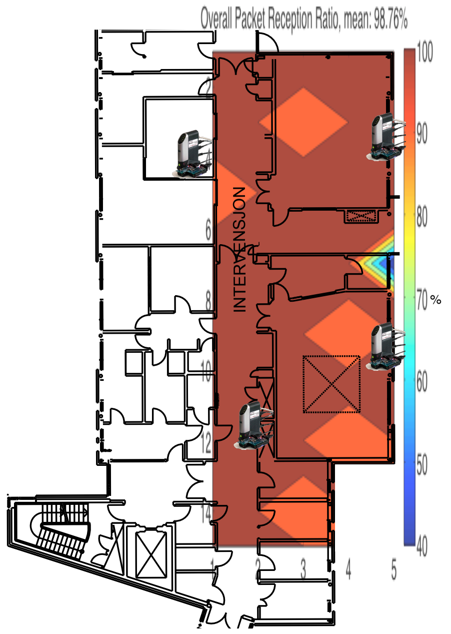
6.5 Result Experiment 3: Base Station Coverage Range

The four base stations were set up to cover an area of approximately $240m^2$. The area was divided into 75 uniformly distributed measurement points to create maps over the packet reception ratio, RSSI, and LQI values for all base stations. All maps created from this simulation are provided in the appendix A.1. Figure 6.2 shows the maps for the overall performance for all base stations combined. For the maps in figure 6.2, for each coordinate, the highest value amongst the different base stations is chosen. E.g. base station 1 receives 99% of the packets sent at coordinate (i,j), while base station 2, 3, and 4 receives 98%, 97%, and 100% respectively, then 100% will be chosen as the overall value for coordinate (i,j). The same procedure is done when creating the overall RSSI and LQI values. When reading all color maps created in experiment 3, note that dark red indicates the best value, and the dark blue indicates the lowest value occurred.

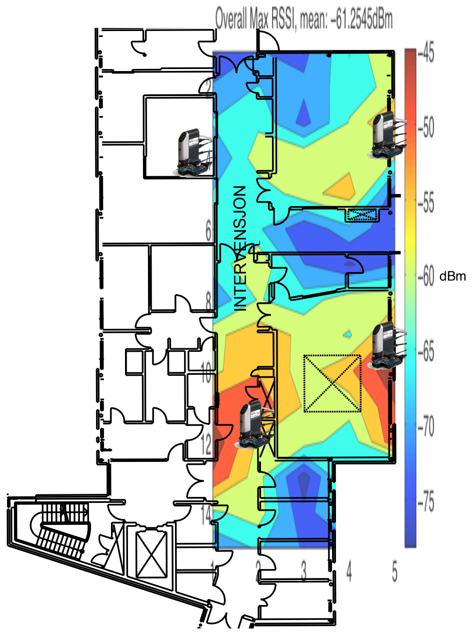
Looking at figure 6.2(a), the packet reception ratio is very high for almost all coordinates, and the total performance is very satisfactory. If an overall lost packet is defined as a packet that was not received by any of the base stations, the calculated packet loss is:

```
Lost 1 Packet(s) After ID: 11180
Lost 1 Packet(s) After ID: 12238
Lost 2 Packet(s) After ID: 13180
Lost 1 Packet(s) After ID: 13235
Lost 1 Packet(s) After ID: 13389
Lost 1 Packet(s) After ID: 13815
Lost 5 Packet(s) After ID: 14242
Lost 1 Packet(s) After ID: 14243
Lost 24 Packet(s) After ID: 14244
```

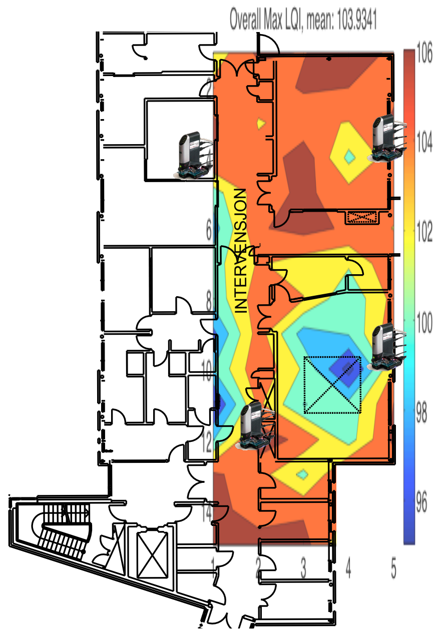
All together, totally 37 packets were lost, which corresponds to 0.97368%. This is extremely low taken into consideration the different positions packets were sent from. Some positions are closer to the base station than others, and a clearer difference between the different coordinates were expected. The above list also indicates that packets usually are not lost consecutively, except from the last occasion where 24 packets were lost in a burst. These packets were lost at the only blue point in figure 6.2(a). The blue point is situated far from base stations 1 and 4, and in between 2 and 3. In addition, there are several walls around blocking for the line of sight, causing many packets to be lost. If the last occasion where 24 packets were consecutively lost can be treated as a special case and be disregarded from the calculation, the packet loss is surprisingly as low as 0.34211%. This indicates that the roaming protocol works well and that combining several receivers will enhance the overall performance.



(a) Overall Packet Loss Ratio

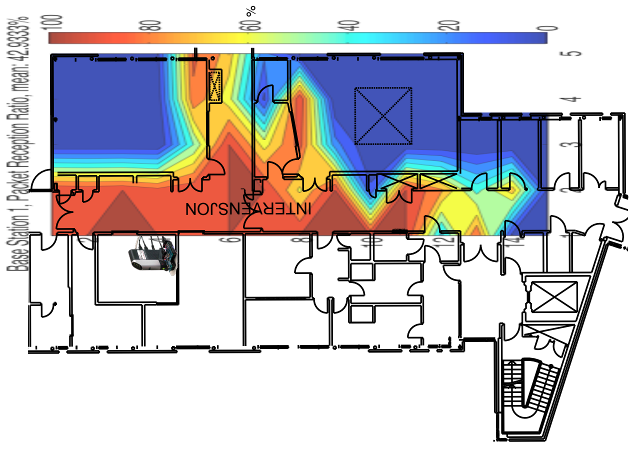


(b) Overall Maximized RSSI

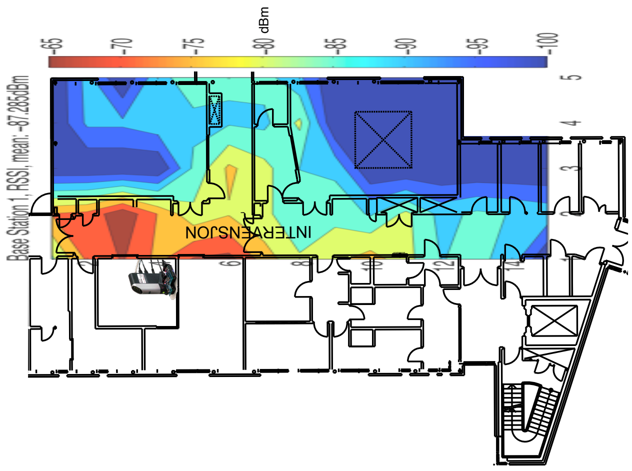


(c) Overall Maximized LQI

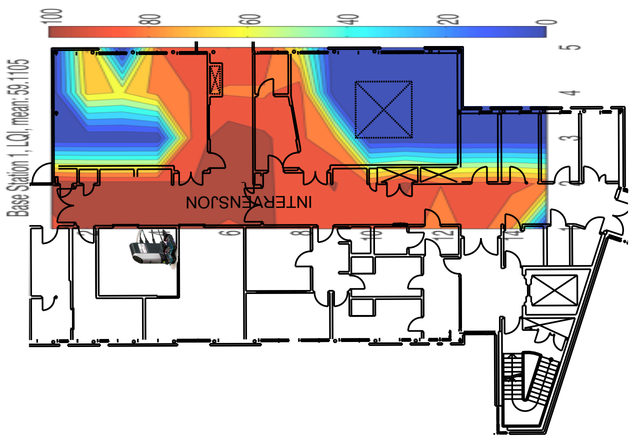
Figure 6.2: All Base Stations



(a) Packet Loss Ratio



(b) RSSI



(c) LQI

Figure 6.3: Base Station 1

Figure 6.3 shows the maps for base station 1. The map for packet reception ratio for base station 1 clearly demonstrates how important line of sight is for packet reception. It is very interesting to see how clearly the packet loss follows the walls. Looking at packet reception ratio for base station 1, figure 6.3(a), and base station 4, figure A.5(a) in the appendix, we can see that base station 4 has a much better performance in the corridor than base station 1. This is probably because base station 1 is placed behind a wall. This indicates that communication between two rooms will be better if a door between them was open.

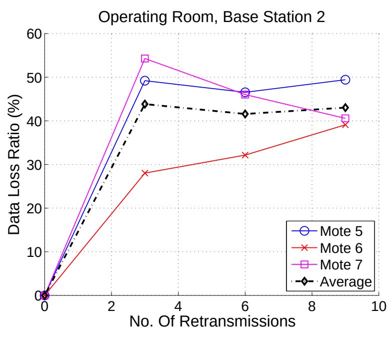
For all base stations, there is a clear correlation between packet reception ratio, RSSI and LQI values.

6.6 Result Experiment 4: Medical Scenarios

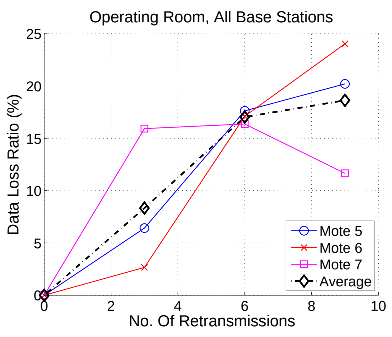
For all medical scenarios, all base stations were operating and had the opportunity to receive all messages if they were in the base station's coverage range.

6.6.1 Operating Room

Figure 6.4 shows the total packet loss from the operating room scenario. Each point in the figure represents the average of 3 separate experiments, each with a duration of approximately two minutes. Figure 6.6(a) shows the packet loss for base station 2 separately, the one that was placed in the laparoscopy room. Figure 6.6(b) is the packet loss when including all base stations as receivers.



(a) Only Base Station 2



(b) All Base Stations

Figure 6.4: Total Packet Loss for Operating Room Scenario

The first thing to notice is the incredibly 0% packet loss for all motes when no acknowledgment and no retransmission is used. Even the base station 2 alone managed to receive all packets that were sent. Further, it is clear that requiring acknowledgment and performing retransmission will decrease the performance. If acknowledgment and retransmission have to be employed, several base stations should be used. In general, figure 6.4 and table A.1 show that using several base stations always yield a better overall system performance.

The RSSI and LQI values for the link between base station 2 and the motes are calculated and provided in table A.2. For all motes, the link quality is almost as high as the maximum achievable.

The numerical simulation results for the operating room scenario is provided in table A.1.

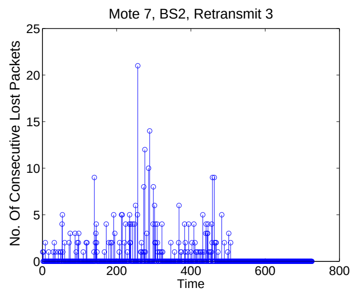
Packet Jitter

Packet jitter is defined as the number of consecutive dropped packets. Figure 6.5 shows packet jittering for the best case and the worst case of packet loss. From table A.1, we can see that the best achieved packet loss ratio (except from those with no loss) is for mote 6 using 3 retransmissions, and accepted by all base stations. The worst case is for mote 7 using 3 retransmissions, for base station 2.

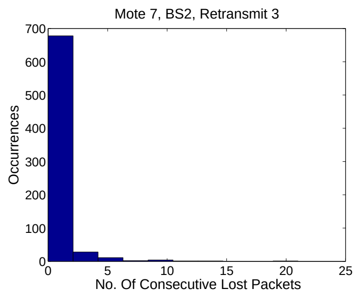
For the best case, the number of consecutive packets lost was never higher than 1. For the worst case, the number could go as high as over 20. If the jitter were very large, we would be concerned that much critical medical data would be lost.

6.6.2 Intensive Care Unit

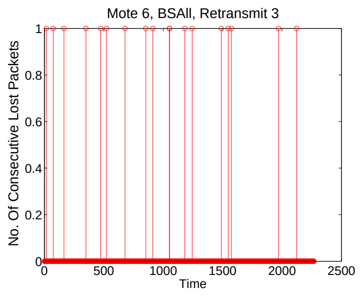
The purpose of this test was to test BWSN's scalability and how it handles a larger number of motes. Figure 6.6 shows the protocols performance in an environment similar to an intensive care unit scenario. Also here, the incredibly, and a bit surprising packet loss of 0% is encountered for the case when no acknowledgement is required. A bit surprising is also the fact that the average performance has not decreased significantly compared to the case with three motes. The overall performance is clearly better when all base stations are used, at least when acknowledgment is required. The test numerical results are provided in A.2.2.



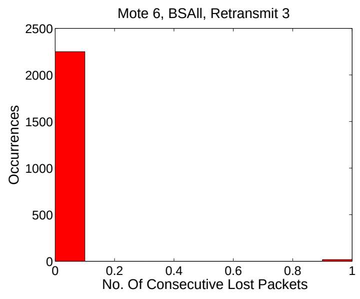
(a) Worst Case



(b) Worst Case Histogram

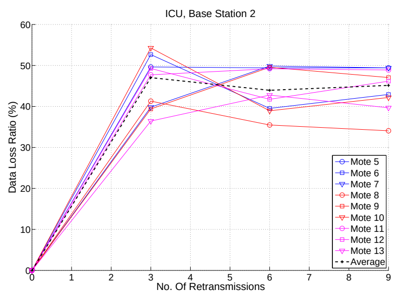


(c) Best Case

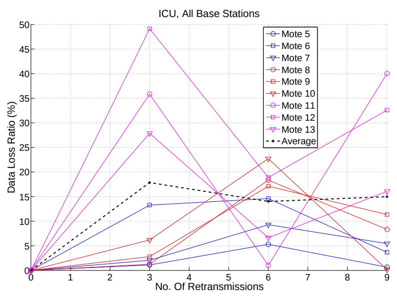


(d) Best Case Histogram

Figure 6.5: Packet Jitter



(a) Only Base Station 2

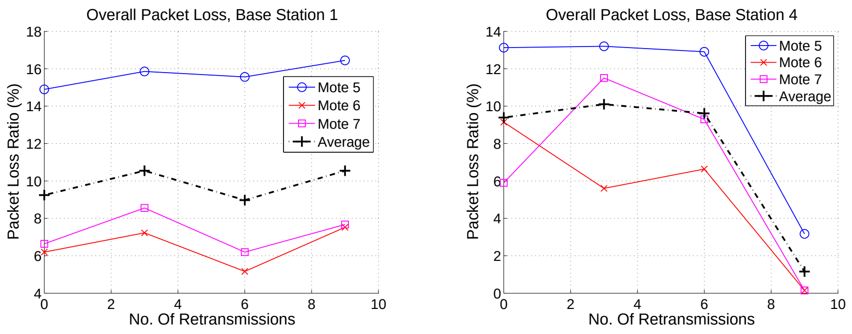


(b) All Base Stations

Figure 6.6: Total Packet Loss for Intensive Care Unit Scenario

6.6.3 Patient Mobility

The mobility test was performed in the corridor, simulating patients being transported from one end of the corridor to the other. Since the test was no longer in the laparoscopy operating room, base station 2 is not that important anymore. Base stations 1 and 4 are now more involved, the figure 6.9 shows the data loss ratio for base station 1 and 4 separately. The overall packet loss at base station 1 is significantly higher than at base station 4.



(a) Base Station 1 (b) Base Station 4

Figure 6.7: Total Packet Loss for Mobility Scenario

For base station 4, the trend is differently than so far seen in the other scenarios. The results show that acknowledgment and retransmissions should be used when the mote is mobile. Figure 6.8 shows the overall performance when all base stations are taken to consideration. By combining many base stations and increasing the number of retransmissions, the average packet loss is less than 1% which is very good for mobile applications.

Since base station 1 and 4 are placed at each end of the corridor, when the motes are approaching one base station, they go further away from the other. This means that something has to be inverted versions of each other, either the packet jitter in time or LQI or RSSI seen from the base stations view. This inversion is actually found in the RSSI value measured by the two base stations. Figure 6.9(a) clearly shows that the RSSI values are inverted versions of each other in the area separated by the green bars. Looking from base station 1, the motes are moving away, towards, and then away again. This means that if packet jitter is plotted with respect to time, it should first increase, decrease, and then increase again. Figure 6.9(b) shows that this actually happens.

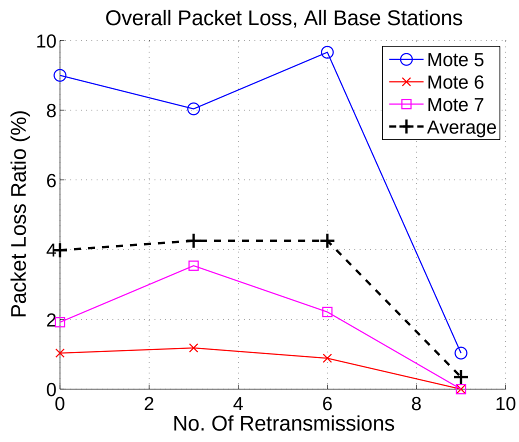
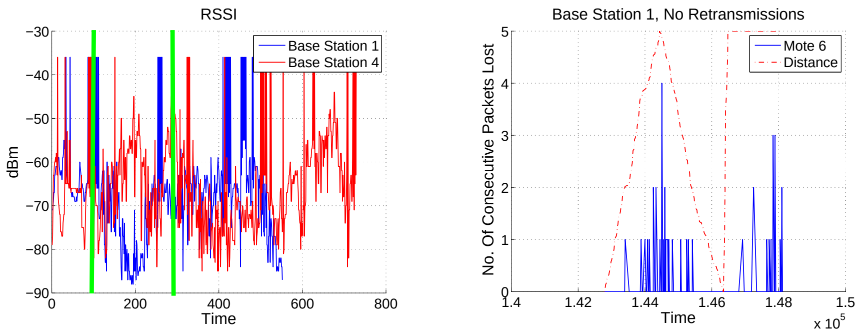


Figure 6.8: Total Packet Loss for Mobility Scenario, All Base Stations Combined



(a) Instantaneous RSSI Values

(b) Packet Jitter in Time

Figure 6.9: Distance Dependent Functions

CHAPTER 7

Discussion

This chapter discusses some of the observations made from the experiments, i.e. unexpected results will be highlighted and suggestions for improvement will be made.

7.1 Synchronization

The attempt of including NetSync in BWSN was widely unsuccessful. More research should have been done before trying to implement NetSync. Even though NetSync could not be used without NetWake, there is still a very big advantage with NetSync. The fact that it is already implemented in TinyOS and provided in Moteiv's TinyOS distribution, makes it still very interesting.

The synchronization requirements in BWSN are not very stringent, and the performance of NetSync is perfectly suitable for the use in BWSN. Other state-of-the-art synchronization protocols offers extremely accuracy, but they often bring a more computationally complex implementation. By further investigating NetSync, it can serve as an implementation model and give inspiration to software solutions in TinyOS. Modules may be reused or rewritten to fit into BWSN.

An implementation can also be made from scratched to serve BWSN and optimized for its needs. Some simple timestamping and calculations of offsets to the synchronization coordinator are maybe good enough to serve BWSN. While measuring the start up procedure, an observation was made.

Recall section **??**. The time stamp t_0 was identical for all runs. This means that the local clock on the motes are not drifting when executing the code, but drifting between different motes occur. A very simple synchronization protocol can be implemented, exchanging some timestamps and create an estimate of the drifts between the different motes' clock. Using this estimate together with an offset of the global time is maybe enough to meet the requirements in BWSN.

7.2 Start Up Procedure

As mentioned in section **??**, the average time it took from sending the a boot up message until the mote is fully operative, was 67.56 ms. Compared to other well know protocols, such as Bluetooth, 67.56ms is ultra-fast. According to [17], the start up time for Bluetooth may be as much as 6 seconds. In medical applications, communication should happen with low latency, so that vital changes in the patient's medical condition can be observed immediately.

However, the results served are based on a best case event. At present time, some weaknesses in the start up procedure are known, but has not been prioritized so far. When the server receives a BootIndicationMsg, it simply set this address (of the distributed address) as taken and reply with a corresponding ConfigureMsg. What happens if the ConfigureMsg is lost? So far, the mote just started up will require a new address, and the server will provide. At the worst case, several addresses will be set as taken, giving a false indication about which motes that exist on the network.

There are several ways to improve the start up procedure. The server should just set the address as reserved, and not taken when sending out a ConfigureMsg. It can require an acknowledge before finally setting the address as taken. Otherwise, the server should at all time be so updated as possible on which clients exist on the network. Therefore, it should regularly perform scans and search for nodes that are supposed to be active. If they are not, the address can be released and provided another mote starting up at a later occasion.

7.3 Base Station Bandwidth

As presented in the results, TOSBase could handle data rate up to approximately 40kbps without problems. When increasing beyond this, the packet loss radio increases exponentially. As observed, the blue LED was blinking rapidly indicating packets being dropped by TOSBase. TOSBase drops packets whenever the queue is full and it has not managed to empty it.

ZigBee offers 250kbps as a theoretical maximum data rate. It is expected that the data rate will be much less in practice, but 40-50kbps is really low. Probably, there is a bottle neck in the system somewhere. The bottle neck may be at TOSBase or at the Universal Asynchronous Receiver/Transmitter (UART) link. TOSBase will drop packets if radio messages are arriving more often that it can transfer messages out of the queue. It can't transfer the messages fast either because it is too computationally week to process fast enough, or the UART can not handle the transferring speed TOSBase wants.

When testing the bandwidth of TOSBase, it was connected to the computer, running a Java application. For Tmote's, the baudrate used in communication with java.comm is 57.6kbaud. When communicating over UART, parity check bits are often used, in addition to start and stop bits. This may have been the bottle neck in the test.

7.4 Base Station Coverage Range

The coverage range of Tmote Sky's connected to Tmote Connect was much better than expected. Testing with four motes to cover an area of approximately $240m^2$ yield very satisfactory results. The overall packet loss was less than 1 %, which is very good for such a big area. In addition, the base stations were placed at edges of the testing area, and if it is assumed that the coverage area is uniformly around each base station, functional coverage is even larger.

The maps created from the base station coverage testing, appendix A.1, provide very interesting information. It tells a lot about the effects of line of sight and how the packet loss ratio, RSSI and LQI will behave nearby thick walls or heavy obstacles.

Such maps are also useful for optimization placements of the base stations. As can be observed in A.1, base station 4 actually covers the whole corridor itself, and the optimal placement would have been to place a base station in the middle of the corridor instead of two at each end.

Further, by looking at the packet reception ration maps compared to the corresponding RSSI and LQI maps, we can further understand the correlation between these factors. At least were packet loss is high, RSSI and LQI are also lowered.

7.5 Medical Scenarios And Roaming

All tests performed for the medical showed very promising packet loss ratio. Overall, the packet loss for both the operation and intensive care unit scenarios were above expectations. Even for the case of 9 different motes sending at the same time, no packet loss was encountered if no acknowledge was required. The reason is probably the static spreading scheme described in the implementation chapter. All motes are starting to send at different time, and for TOSBase it handled the load well when packets were not coming almost simultaneously.

For both the intensive care unit scenario and the operation scenario the links to the base station was strong, in deed very strong. The RSSI and LQI values calculated confirmed this. In the case of 9 motes, the data rate was 19,2kbps all together excluding overhead. Including overhead, this was 22.857kbps. From the bandwidth test of TOSBase, this is a ratio it should handle easily.

Further, the packet jitter was also low, which is very desirable in medical applications. If 100 packets are lost over a longer period of time, this is not as worse as loosing them in a burst. If small parts of the medical data is lost, some interpolation or other recovery techniques can be applied. If a burst of data is lost, no recovery can be made.

It was also very promising to see the roaming performing so well. Data delivery was very very high, and all tests showed that the performance was always better when using several base stations. Compared to both ZigMed and CodeBlue, BWSN performed significantly better. In CodeBlue, for the case of 10 motes sending a total data rate of slightly above 20kbps, the average reception ratio was below 60% when several receivers were implemented. For the case of single receiver, CodeBlue had ten sensors sending 21kbps, and the average reception ratio was slightly above 40%. BWSN used the same data rate to achieve much much lower packet loss ratio.

The scenario testing clearly indicated that acknowledge and retransmissions should not be done in situations when a strong RF link already exists. By acknowledging packets, the air will be saturated and in turn lead to dropped packets. All in all, the roaming protocol was the main reason for the successfully high data rate, and it has definitely been a positive add-on for the BWSN protocol. It makes the communication scheme robust and optimizes data throughput. By combining several receivers, the packet loss ratio was minimized to make the communication reliable.

CHAPTER 8 Conclusion and Future Work

In this thesis, a cross-layer communication protocol was customized for the BWSN project. Several design issues were presented, and had to be considered. Extensive programming was performed, and the worked required much testing.

A roaming protocol was implemented, turning out to be a one of the main reasons why the protocol works so well. When testing the protocol in real medical scenarios, the performances were above expectations. Compared to previous work, the protocol performs much better when it comes to data throughput, which was one of the main criteria for optimization.

The conclusions from the medical tests showed that for communication where good links exist, use of acknowledge and retransmission will just decrease the performance. In bad links or in situations where the environment is changing, for instance when a node is in mobility, acknowledge should be used to increase the performance. In all scenarios, all best cases always showed less than 1% of packet loss, even when the mote was moving.

The roaming protocol optimized the system for its purposes, but it also introduces several opportunities. By using a "selective-broadcast" -like approach, the roaming protocol worked perfectly well in the medical scenarios. In introduced no additional protocol overhead, no base station handover switching and no additional processing had to be done by the motes in the network.

The natural further step in BWSN is to extend the server possibilities for more control and feedback of the network. Some sort of resource allocating and network monitoring should maybe be implemented. Later, energy conservation algorithms should maybe also be considered.

BIBLIOGRAPHY

[1] Muneeb Ali, Umar Saif, Adam Dunkels, Thiemo Voigt, Kay Römer, Koen Langendoen, Joseph Polastre, and Zartash Afzal Uzmi. Medium access control issues in sensor networks. *SIGCOMM Comput. Commun. Rev.*, 36(2):33–36, 2006.

[2] Ilangko Balasingham. Practical use of biomedical sensors and wireless networks. Presentation, 2006.

[3] Paolo Baronti, Prashant Pillai, Vince Chook, Stefano Chessa, Alberto Gotta, and Y. Fun Hu. Wireless Sensor Networks: a Survey on the State of the Art and the 802.15.4 and ZigBee Standards. Technical Report C.2 COMPUTER.COMMUNICATION NETWORKS, Universiy of Bradford and University of Pisa, May 2006.

[4] S. Basagni. Distributed Clustering for Ad Hoc Networks. In *Proc. Int'l. Symp. Parallel Architectures, Algorihms, and Nets.*, Perth, Australia, June 1999.

[5] BWSN. Biomedical Wireless Sensor Network Project, 2007. http://www.bwsn.net.

[6] Alberto Cerpa and Deborah Estrin. ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. *IEEE Transactions on Mobile Computing*, 3(3):272–285, 2004.

[7] Bor-rong Chen, Kiran-Kumar Muniswamy-Reddy, and Matt Welsh. Ad-hoc multicast routing on resource-limited sensor nodes. In *REALMAN '06: Proceedings of the second international workshop on Multi-hop ad hoc networks: from theory to reality*, pages 87–94, New York, NY, USA, 2006. ACM Press.

[8] Moteiv Corporation. Tmote sky. http://www.moteiv.com/products/tmotesky.php.

[9] Dennis Cox, Emil Jovanov, and Aleksandar Milenkovic. Time synchronization for zigbee networks. In *System Theory, 2005. SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium on, Vol., Iss., 20-22 March 2005*, pages 135–138, Huntsville, USA, 2005. IEEE.

[10] Crossbow Technology, Inc. MICA2 Series (MPR4x0). http://www.xbow.com/Products/productdetails.aspx?sid=174.

[11] Crossbow Technology, Inc. MICAz ZigBee Series (MPR2400). http://www.xbow.com/Products/productdetails.aspx?sid=164.

[12] Shuguang Cui, Ritesh Madan, Andrea Goldsmith, and Sanjay Lall. Joint routing, mac, and link layer optimization in sensor networks with energy constraints. *Communications, 2005. ICC 2005. 2005 IEEE International Conference*, 2:725–729, May 2008.

[13] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Oper. Syst. Rev.*, 36(SI):147–163, 2002.

[14] TinyOS Community Forum. http://www.tinyos.net.

[15] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149, New York, NY, USA, 2003. ACM Press.

[16] David Gay, Philip Levis, David Culler, and Eric Brewer. *nesC 1.1 Language Reference Manual*. UC Berkeley WEBS Project, Berkeley, USA, May 2003. http://nescc.sourceforge.net/.

[17] Mats Skogholt Hansen. A Comparative Study of WLAN, Bluetooth and ZigBee for Bio-Sensor Networks. Project, 2005.

[18] Mats Skogholt Hansen and Stig Støa. Practical Evaluation of IEEE 802.15.4/ZigBee Medical Sensor Networks. Master's thesis, Norwegian University of Science and Technnology, Trondheim, 2006.

[19] Dr. Chew Yong Huat. Cellular systems. Lecture Notes: EE5401 Cellular Mobile Communications - National University of Singapore. http://www1.i2r.a-star.edu.sg/ chewyh/.

[20] IEEE Standards. *IEEE 802 Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs).* IEEE Computer Society/The Institute of Electrical and Electronics Engineers, Inc, New York, USA, October 2003.

[21] Texas Instruments. Data Sheet CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver, 2006. http://www.chipcon.com.

[22] Jorjeta G. Jetcheva and David B. Johnson. Adaptive demand-driven multicast routing in multi-hop wireless ad hoc networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 33–44, New York, NY, USA, 2001. ACM Press.

[23] Meng Jiang. Tele-cardiology Sensor Networks for Remote ECG Monitoring. Master's thesis, Rochester Institute of Technology, New York, 2006.

[24] Koen Langendoen, (Edited by H.Wu and Y.Pan). Medium Access Control In Wireless Sensor Networks. In *Medium Access Control in Wireless Networks, Volume II: Practice and Standards.* Nova Science Publishers, 2007.

[25] Konrad Lorincz and Matt Welsh. Motetrack: A robust, decentralized approach to rf-based location tracking. In *LoCA*, pages 63–82, 2005.

[26] David Malan, Thaddeus Fulford-Jones, Matt Welsh, and Steve Moulton. An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care. In *International Workshop on Wearable and Implantable Body Sensor Networks*, April 2004.

[27] Miklos Maroti, Branislav Kusy, Gyula Simon, and Akos Ledeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, New York, NY, USA, 2004. ACM Press.

[28] Tommaso Melodia, Mehmet C. Vuran, and Dario Pompili. The State of the Art in Cross-layer Design for Wireless Sensor Networks. In *Proceedings of EuroNGI Workshops on Wireless and Mobility. Springer Lecture Notes in Computer Science 3883*, Como, Italy, July 2005.

[29] Aleksandar Milenkovic, Chris Otto, and Emil Jovanov. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Computer Communications (Special issue: Wireless Sensor Networks: Performance, Reliability, Security, and Beyond)*, 29(13 14):2521–2533, 2006.

[30] David L. Mills. Internet time synchronization: The network time protocol. In *Zhonghua Yang and T. Anthony Marsland (Eds.), Global States and Time in Distributed Systems.* IEEE Computer Society Press, 1994.

[31] H. S. Ng, M. L. Sim, C. M. Tan, and C. C. Wong. Wireless technologies for telemedicine. *BT Technology Journal*, 24(2):130–137, 2006.

[32] Sajdl Ondrej, Bradac Zdenek, Fiedler Petr, and Hyncica Ondrej. ZigBee Technology and Device Design. In *Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)*, Brno, Czech Republic, 2006. IEEE Computer Society.

[33] Chris Otto, Aleksandar Milenkovic, Corey Sanders, and Emil Jovanov. System architecture of a wireless body area sensor network for ubiquitous health monitoring. *Journal of Mobile Multimedia*, 1(4):307–326, 2006.

[34] Telos Platform. http://www.moteiv.com/products-reva.php.

[35] Joseph Polastre. TinyOs Mail Archive. https://mail.millennium.berkeley.edu/pipermail/tinyos-help/2006-June/017051.html.

[36] Joseph Polastre, Jonathan Hui, Philip Levis, Jerry Zhao, David Culler, Scott Shenker, and Ion Stoica. A unifying link abstraction for wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 76–89, New York, NY, USA, 2005. ACM Press.

[37] The Mobihealth Project. Innovative gprs/umts mobile services for applications in healthcare. http://www.mobihealth.org/.

[38] Srajan Raghuwanshi and Amitabh Mishra. A self-adaptive clustering based algorithm for increased Energy-efficiency and Scalability in Wireless Sensor Networks. *Vehicular Technology Conference*, 5:2921–2925, October 2003.

[39] Sanjay Shakkottai, Theodore S. Rappaport, and Peter C. Karlsson. Cross-layer design for wireless networks. *Communications Magazine, IEEE*, 41(10):74–80, 2003.

[40] Victor Shnayder, Bor rong Chen, Konrad Lorincz, Thaddeus R. F. Fulford Jones, and Matt Welsh. Sensor networks for medical care. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 314–314, New York, NY, USA, 2005. ACM Press.

[41] SMART. Smart: Scalable medical alert and response technology. http://smart.csail.mit.edu/.

[42] Åsmund Sandvand. Bwsn sensor message format v1, June 2007. http://www.bwsn.net.

[43] Stig Støa. Sensornettverk for medisinsk behandling. Project, 2005.

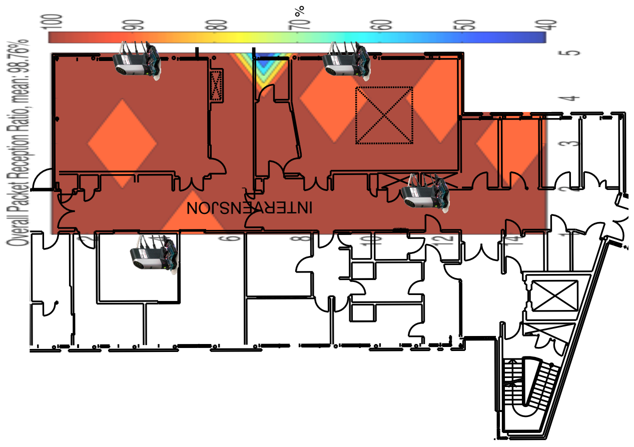
[44] Weilian Su and Tat L. Lim. Cross-layer design and optimization for-wireless sensor networks. In *SNPD-SAWN '06: Proceedings of the Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'06)*, pages 278–284, Washington, DC, USA, 2006. IEEE Computer Society.

[45] The CodeBlue Project. CodeBlue: Wireless Sensor Networks for Medical Care. http://www.eecs.harvard.edu/ mdw/proj/codeblue/.

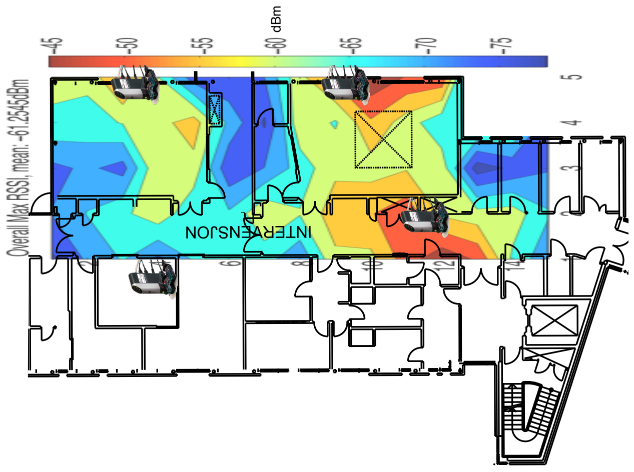
[46] Loc Tan Vo. A Prestudy on the IEEE 802.15.4 Based Medical Sensor Networks. Project, 2007.

APPENDIX A Simulation Results

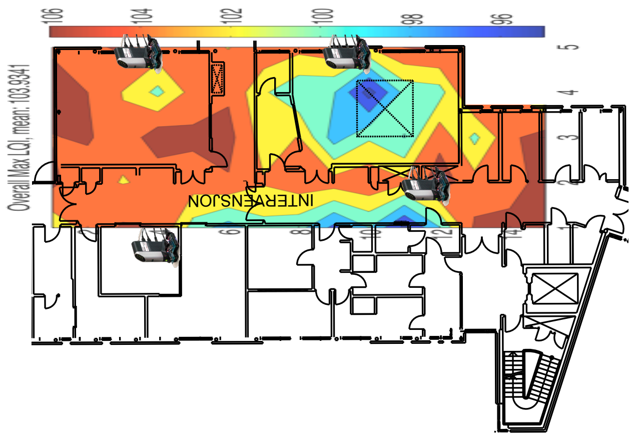
A.1 Base Station Coverage



(a) Overall Packet Loss Ratio

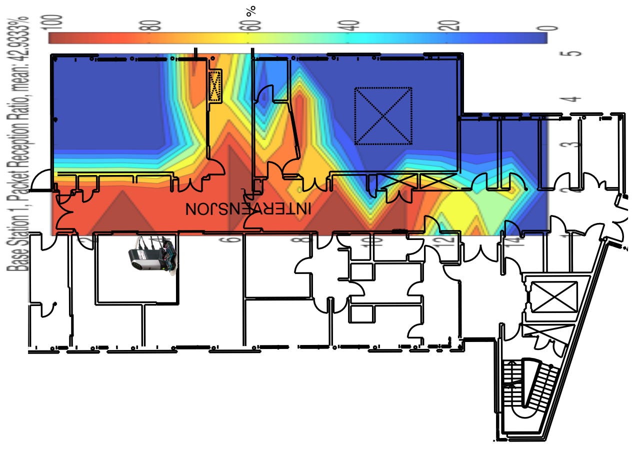


(b) Overall Maximized RSSI

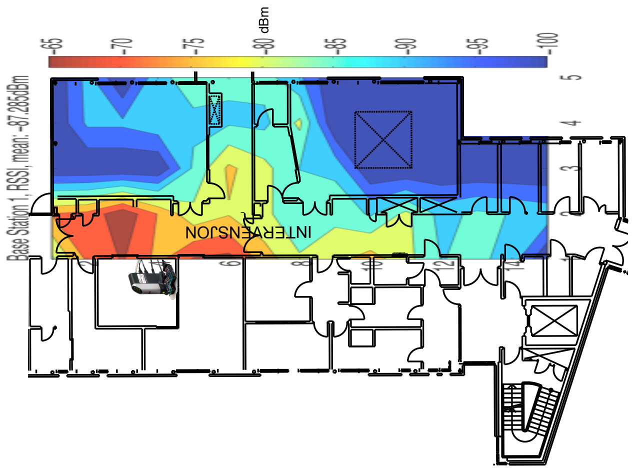


(c) Overall Maximized LQI

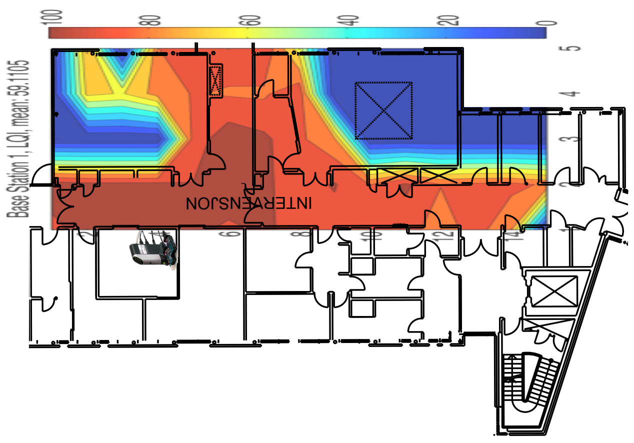
Figure A.1: All Base Stations



(a) Packet Loss Ratio

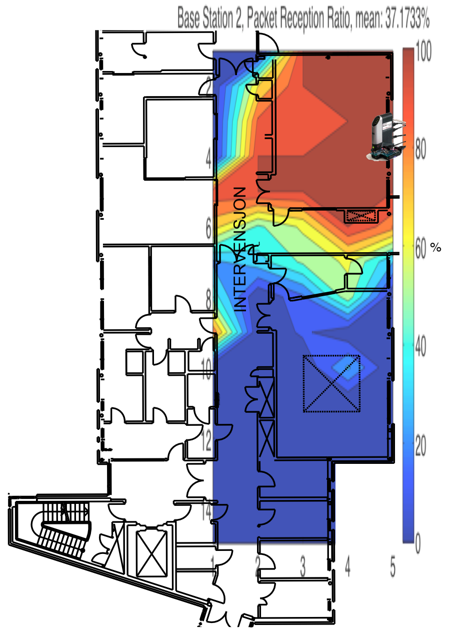


(b) RSSI

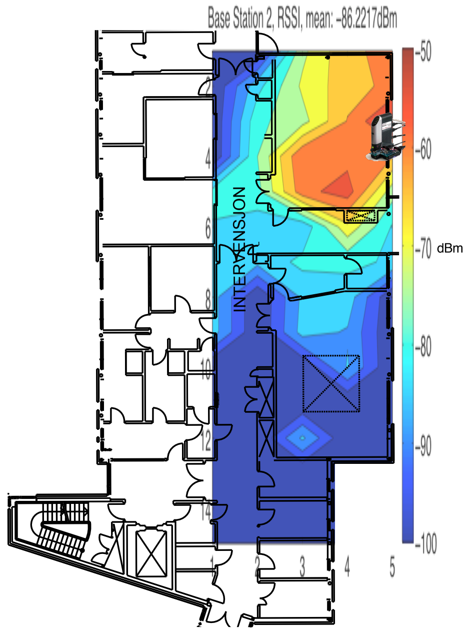


(c) LQI

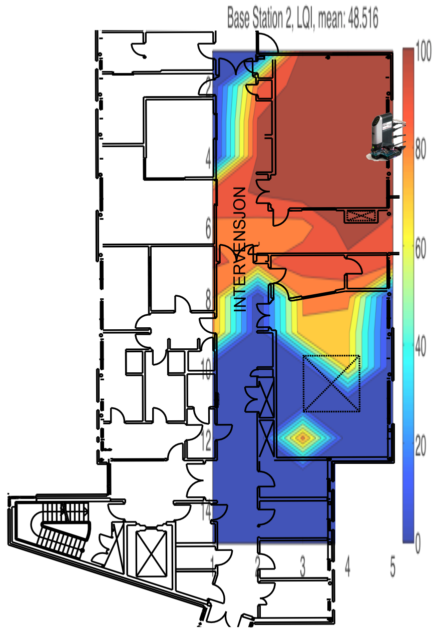
Figure A.2: Base Station 1



(a) Packet Loss Ratio

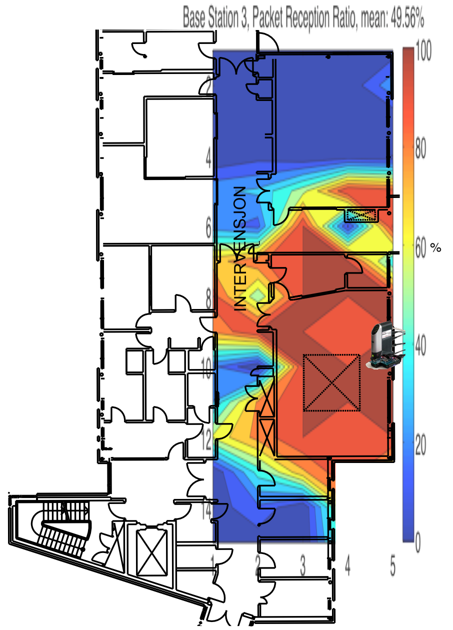


(b) RSSI

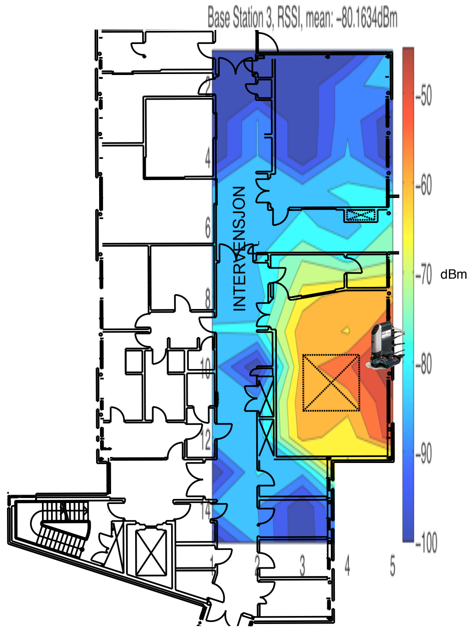


(c) LQI

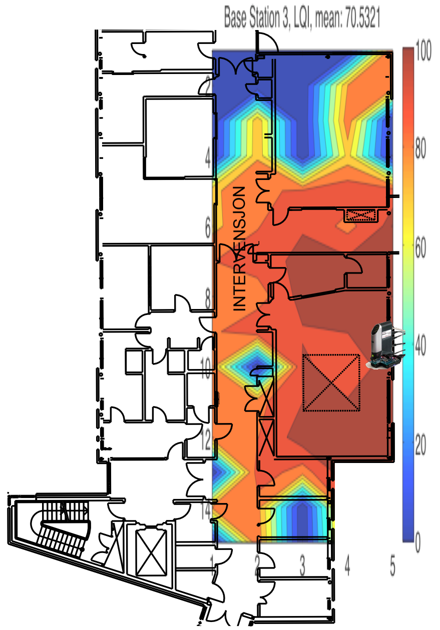
Figure A.3: Base Station 2



(a) Packet Loss Ratio

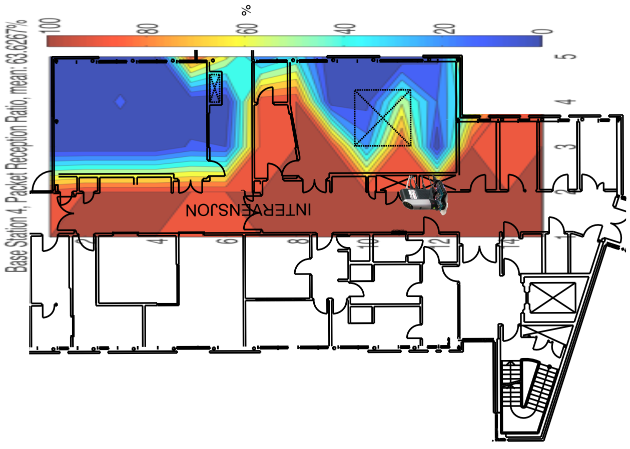


(b) RSSI

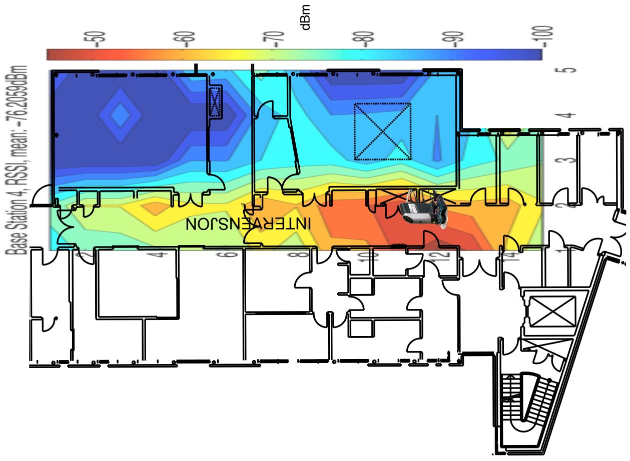


(c) LQI

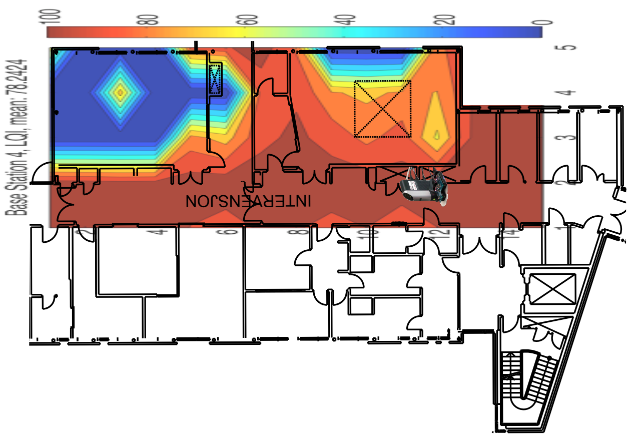
Figure A.4: Base Station 3



(a) Packet Loss Ratio



(b) RSSI



(c) LQI

Figure A.5: Base Station 4

A.2 Result Experiment 4: Medical Scenarios

A.2.1 Operating Room Scenario

# Retrans.	**Mote 5**		**Mote 6**		**Mote7**	
	Base Station 2	All Base Stations	Base Station 2	All Base Stations	Base Station 2	All Base Stations
0	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0
3	49.189 %	6.4159 %	28.024 %	2.6549 %	54.277 %	15.929
6	46.534 %	17.625 %	32.153 %	17.109 %	46.018 %	16.372
9	49.410 %	20.206 %	39.086 %	24.041 %	40.560 %	11.652

Table A.1: Packet Loss For Operating Room Scenario

	Mote 5		**Mote 6**		**Mote7**	
	RSSI	**LQI**	**RSSI**	**LQI**	**RSSI**	**LQI**
Max	-41 dBm	108	-48 dBm	108	-45dBm	108
Min	-44 dBm	101	-56 dBm	101	-48dBm	103
Average	-42.8053 dBm	105.2957	-51.4115 dBm	105.6681	-45.0472 dBm	106.5693

Table A.2: Link Quality Between Motes And Base Station 2

A.2.2 Intensive Care Unit Scenario

#Retr.	Mote5		Mote6		Mote7	
	BS2	All BSs	BS2	All BSs	BS2	All BSs
0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.1062	49.631	13.274	52.655	2.0649	39.823
6	5.3097	49.410	14.602	39.528	9.2920	49.852
9	0.66372	49.410	3.6873	42.920	5.4572	49.410

Table A.3: Packet Loss Mote 5, 6, and 7

#Retr.	Mote8		Mote9		Mote10	
	BS2	All BSs	BS2	All BSs	BS2	All BSs
0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.2537	41.298	2.8024	39.381	6.1947	54.277
6	18.363	35.472	17.109	49.558	22.714	38.938
9	8.3333	34.071	11.357	47.050	0.14749	42.183

Table A.4: Packet Loss Mote 8, 9, and 10

#Retr.	Mote11		Mote12		Mote13	
	BS2	All BSs	BS2	All BSs	BS2	All BSs
0	0.0	0.0	0.0	0.0	0.0	0.0
3	35.841	47.714	49.115	49.263	27.876	36.431
6	1.0324	49.189	18.879	41.740	6.6372	42.773
9	40.044	48.894	32.596	46.165	16.077	39.676

Table A.5: Packet Loss Mote 11, 12, and 13

APPENDIX B Source Code

The source code consisting of nesC files, JAVA files and Matlab scripts are all bundled and provided at:

`http://folk.ntnu.no/loctan/master/master.zip`

The nesC files are TinyOs files to be used on the motes. JAVA files are basically the files describing the server's behavior, while the Matlab scripts contain SQL queries and data processing to visualize the test results.