E-SENSE: A WIRELESS SENSOR NETWORK TESTBED AND SYSTEM FOR MONITORING INBUILDING ENVIRONMENTS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Berk Berker

August, 2008

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. İbrahim Körpeoğlu (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Ali Aydın Selçuk

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Adnan Yazıcı

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

E-SENSE: A WIRELESS SENSOR NETWORK TESTBED AND SYSTEM FOR MONITORING INBUILDING ENVIRONMENTS

Berk Berker
M.S. in Computer Engineering
Supervisor: Assist. Prof. Dr. İbrahim Körpeoğlu
August, 2008

Wireless sensor networks consist of small, smart and battery-powered devices suitable for widespread deployment to monitor an environment by taking physical measurements. Wireless sensor nodes are deployed over an area in a random manner. They need to self-establish a wireless multi-hop network and routing paths from all sensor nodes to a central base station. In this thesis, we present our E-Sense system, a wireless sensor network testbed consisting of MICA2 sensor nodes which can be used to monitor an indoor environment like office buildings and homes. The testbed can be accessed through the Internet and provides a web-based interface to the sensor network. The users of the network can be located at any point in the Internet. Via the web based interface, the users can submit various types of queries to the sensor network and get the replies including the physical measurement results.

The E-Sense system also includes a distributed and energy-aware routing protocol that we designed and implemented. The protocol aims efficient and balanced usage of energy in the sensor nodes to prolong the lifetime of the network. The routing protocol is based on a many-to-one routing tree where each node independently determines its next parent depending on the values of RSSI (Received Signal Strength Indicator). The protocol can also adjust the transmit power to further decrease the energy spent in each sensor node. The testbed will be useful for experimental studies at both application and network levels.

Keywords: Sensor Networks, Environment Monitoring, Multihop Routing, Energy Awareness, RSSI, Transmit Power Adjustment.

ÖZET

E-SENSE: KAPALI ALANLARDA ORTAMI GÖZLEME AMAÇLI KABLOSUZ ALGILAYICI AĞI TEST ALANI VE SİSTEMİ

Berk Berker
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. İbrahim Körpeoğlu
Ağustos, 2008

Kablosuz algılayıcı ağları küçük, akıllı ve pille çalışan cihazlardır ve bu cihazlar, geniş alanlara yayılabilir olup, ortamları fiziksel olçümler alarak gözleyebilirler. Kablosuz algılayıcılar rastgele bir şekilde bir alana yerleştirilirler. Bu algılayıcılar aralarında çok-sekmeli bir ağ oluştururlar ve bütün algılayıcılar baz istasyonuna doğru rota oluştururlar. Bu tezde MICA2 algılayıcılarından oluşan, ofis ve ev gibi kapalı alanları gözleme amaçlı kullanılan kablosuz algılayıcı ağı test alanı E-Sense sistemi önerilmektedir. Bu test alanı İnternetten ulaşılabilir olup algılayıcı ağına web tabanlı arabirim sağlar. Web tabanlı arabirim aracılığıyla kullanıcılar, algılayıcı ağına istek gönderebilir ve bu isteklerin cevaplarını fiziksel ölçümler de dahil olmak üzere görebilir.

E-Sense sistemi ayrıca bizim tasarlayıp uygulamasını yazdığımız dağıtık ve enerjiyi dikkate alan rota protokolü içerir. Protokolün amacı enerjinin algılayıcılarda verimli ve dengeli kullanılması ve ağın ömrünü uzatmaktır. Rota protokolü çok düğümden bir düğüme doğrudur ve ağaç tabanlıdır. Her düğüm bağımsız olarak sinyal gücü değerlerine göre ağaçtaki ata düğümünü belirler. Protokol ayrıca harcanan enerjiyi daha da azaltmak için algılayıcılardaki iletim gücünü ayarlayabilir. Test alanı, uygulama ve ağ seviyelerindeki deneysel çalışmalar için de faydalı olacaktır.

Anahtar sözcükler: Sensör Ağları, Ortamı Gözleme, Rota Belirleme, Sinyal Gücü, İletim Gücü Ayarlama.

Acknowledgement

I would like to thank my supervisor Assist. Prof. Dr. İbrahim Körpeoğlu for his supportive approach and constructive ideas throughout my master's study. His continuous concern made me encourage to study harder. I learned a lot under his supervision of this thesis.

I would also like to thank Assist.Prof.Ali Aydın Selçuk and Prof.Adnan Yazıcı for accepting to spend their valuable time for evaluating my thesis.

I would like to express my gratitude to Scientific and Technical Research Council of Turkey (TÜBİTAK, BİDEB) for their financial support during my master's study.

We thank The Scientific and Technological Research Council of Turkey for supporting this work with project EEEAG-104E028.

I would like to thank my office friends for their support and feedbacks during my master's study.

I would like to thank my aunt and my cousin for accommodating me through my master's study. Their presence made this study more enjoyable.

Last but not least I am grateful to my family for their continuous love, trust and support during my master's study. They did their best to raise me well and gave everything they had. Their encouragement made me believe that I can achieve everything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Advances in hardware technology and electronics have led scientists to develop small, smart and low-powered sensor nodes. A typical sensor node consists of a small radio unit providing a short range wireless communication, a sensing unit for sensing the physical phenomenon, and a data processing unit. The sensor nodes are densely deployed close to the phenomenon or inside the phenomenon. Deployment to regions like buildings, offices, homes, and person areas are possible. The purpose of the sensors are to sense, monitor the environment, and report the results to the base station for further processing [1].

Wireless sensor networks may consist of different types of sensors such as thermal, visual, seismic, and infrared sensors.

There are various applications for wireless sensor networks. Example applications include:

- Home: For self-automation of light and temperature.
- Military: For intrusion detection, battle damage assesment, chemical gas measurement.
- Health: For monitoring doctors and patients in hospital.

- Public: For vehicular movement on roads, fire detection in forests, flood detection.

The area of wireless sensor networks presents many challenges. These are [2]:

- Application specific: In wireless sensor networks, there are many different applications scenarios possible. It is unlikely to be written general purpose applications for wireless sensor networks, therefore we should consider the goals and requirements of the application while designing it for wireless sensor networks.
- Environment interaction: The data rate of the network may be low over a large time scale, but in some occasions like having to reply many queries, there may be some bursty traffic.
- Scale: Wireless sensor networks are tend to scale large numbers, so we should consider solutions that apply for large scale.
- Energy: Energy is a scarce source in wireless sensor networks. Since sensor nodes are mostly battery-powered, we need to prolong the life time of a sensor node which has a deep impact on a wireless sensor network's life time.
- Self configurability: The network traffic and energy trade-offs could require configurable solutions. A sensor node should adjust itself for dynamically changing conditions.

The deployed sensors report their measured data to the base station, also called as the sink node. The reporting may be done periodically or on demand, when a user submits a query to obtain the measured data. The sink node is connected to a data processing machine for further evaluation of the sensed data values.

In order for the sensed data to reach the sink node, the sensors must construct a route to the sink node. The routing problem in wireless sensor networks is a

wide research area. There are several design issues for multihop routing algorithms. These are node deployment, minimizing the energy consumption without losing accuracy, fault tolerance, connectivity, coverage, scalability and transmission media [3]. There are various multihop path construction algorithms in the literature.

Energy consumption is one of the most important problems in wireless sensor networks. In traditional ad hoc networks, preserving energy is not the essential part for MAC protocols and network-layer protocols, thus protocols and algorithms for traditional ad hoc networks does not effectively apply for wireless sensor networks. The energy source of handling wireless communication, sensing the phenomenon, and processing the data is a small battery cell. MAC protocols must have low duty cycle, and routing protocols must be designed to minimize the energy consumption.

Low-powered batteries on sensor nodes are likely to die while functioning. The topology of the sensor network probably changes with the death of each node. One must account these changes and design the algorithm to adapt them.

As a part of this thesis, we propose a multihop routing protocol that considers energy efficiency. Our algorithm considers:

- Received signal strength values for estimating the power costs of the links between the sensor motes.
- Adjusting the transmission power to save energy, thus extending the lifetime of each sensor.

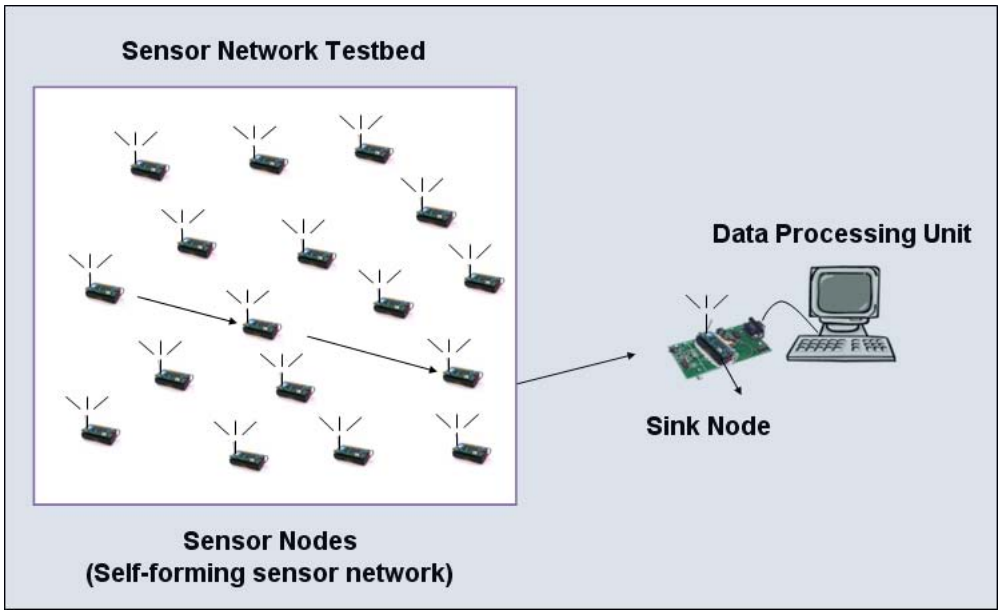


Figure 1.1: Wireless sensor network testbed.

The domain of wireless sensor networks is still young. Researchers are keen to develop wireless sensor network testbeds and multihop network algorithms. Devised algorithms were usually tested in simulated environments. Simulations are good at comparing various algorithms with each other to see which one outperforms the others under certain conditions. However, this does not mean that the algorithm will work the same way on a specific sensor hardware and on a real network. No doubt, real hardware implementations of multihop algorithms are more convincing.

Our project is to design and implement a complete wireless sensor network testbed with real hardware. This testbed is used for sensing the environment. Our project has two goals. The first is to create a working system in which the most essential part is enabling users to access and query the sensed data from any sensor node via the Internet. The second one is to implement an energy-efficient multihop routing algorithm for wireless sensor networks and make it functional with a set of sensors deployed randomly over an area.

In our implementation users can submit queries and see the textual and graphical results derived from the data collected by the sensor network. The results of the queries include the location of a sensor, the temperature around a sensor, the light and the sound amplitude around a sensor, and the remaining battery power of a sensor. Users are not aware of anything except the query submission and the feedbacks they receive from the browser. The system maintenance is simple. The administrator only needs to change the dead batteries with the new ones to keep the system running. The ease of use of this system enables an average Internet user to be able to interact with a wireless sensor network. The provided abstraction helps the user not to deal with the internal parameters of the wireless sensor network.

In Chapter 2, we start with the previous works about proposed multihop algorithms, using RSSI as a distance estimation in indoors, and the transmit power concept. Multihop algorithms are the ones that were implemented for real sensors. Chapter 3 describes what kind of sensor hardware and software was used in the system, and the programming language for sensor networks, nesC, is briefly explained. Chapter 4 presents the E-Sense system, focusing on the design details and the components. Chapter 5 gives information about the multihop algorithm, along with the implementation details. The deployment of the sensors and the conducted experiments are explained in Chapter 6. Finally the thesis ends with conclusions and future works in Chapter 7.

Chapter 2

Background and Related Work

In this chapter, we first mention about some multihop routing protocols designed for wireless sensor networks which are applicable to real sensor nodes.

A multihop routing protocol must be designed in a way such that it meets the requirements of the application. For example; for real time applications, the multihop routing protocol must consider the end-to-end delay and jitter to meet the deadlines. Our protocol's primary objective is to conserve energy in order to extend the lifetime of the first dying node. In other words, we try to maintain all the nodes functional as long as possible.

Our multihop routing protocol takes advantage of the adjustment of transmit powers for conserving more energy. We decrease the transmit power up to a level at which a node can still send its data successfully to its selected parent. To do that, distances between nodes need to be known or estimated. For a good estimation of the distance, we use the metric RSSI (Received Signal Strength Indicator).

After describing some of the already proposed multihop routing protocols, we focus on the effectiveness of RSSI for estimating the distance and touch upon the previous studies on transmit power adjustment.

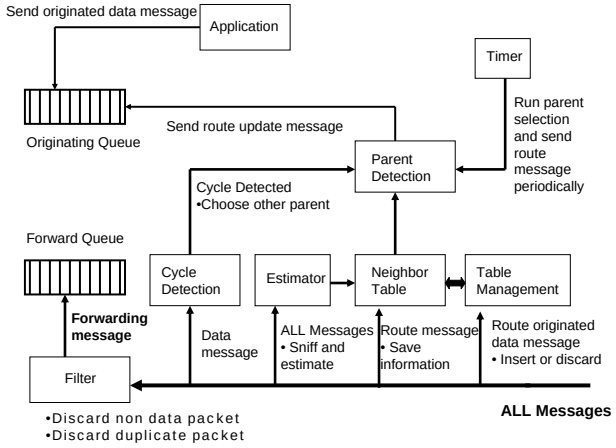


Figure 2.1: Mintroute multihop message flow chart.

2.1 Multihop Routing Protocols

Various multihop routing protocols and algorithms have been proposed in the literature. Some of them were implemented and ported to real sensor platforms while the others were only tested in simulation environments.

One of the protocols that were implemented for real sensors was MintRoute [4]. It was implemented for the TinyOS operating system (see Chapter 3), which is the most popular operating system for wireless sensor networks. The authors focused on the data-collection (many-to-one) routing scenario. MintRoute is a distributed distance-vector based protocol. Each node locally applies parent selection mechanism to route the messages. The routing messages include parent address, estimated routing cost to the sink, and a list of reception link estimations of neighbors. When a routing message is received from a node that already exists in the neighbor table, the corresponding entry is updated. Data packets originating from the node are queued for sending with the parent as the destination. Incoming data packets are selectively forwarded through the forwarding queue with the current parent as the destination address. See Figure 2.1 for an overall illustration.

The routing algorithm accesses the neighbor table and extracts a set of potential parents. A neighbor is selected as a potential parent only if its cost is less than the current cost of the node.

Individual nodes estimate link quality by observing packet success and loss events. Higher-level protocols use these estimations to build routing structures. An alternative approach is to use received signal strength as an estimation of link quality. In our study, we make use of both estimations for determining the link quality.

Our algorithm is a distance vector based algorithm which estimates the link quality using RSSI measurements. RSSI measurements are used to estimate how much power a node spends to reach the base station, and the decision is based on the minimum power cost to the sink node.

Another multihop routing protocol implemented for TinyOS is AODV. As a well known fact, AODV is initially designed for mobile ad hoc networks and it provides unicast, broadcast and multicast communication in ad hoc networks. AODV initiates route discovery whenever a route is needed by a source node or whenever a node wishes to join a multicast group. Routes are maintained as long as they are needed by the source node as long as the multicast group exists, and the routes are always loop-free through the use of sequence numbers. AODV nodes maintain a route table in which next hop routing information for destination nodes is stored [5, 6].

The version implemented for wireless sensor networks differs from the original in various aspects. Route reply messages are only generated by the destination, while in the original AODV the intermediate nodes may send route reply messages as well, if they have a route to the destination. Routes never expire whereas they would expire in the original one if the route is not being used for a period of time. Whenever the path between source and the destination is disconnected, route errors are generated by the uplink nodes [7, 8].

In our protocol, the path is not established on demand, instead we determine the paths proactively by exchanging the control messages over some time period.

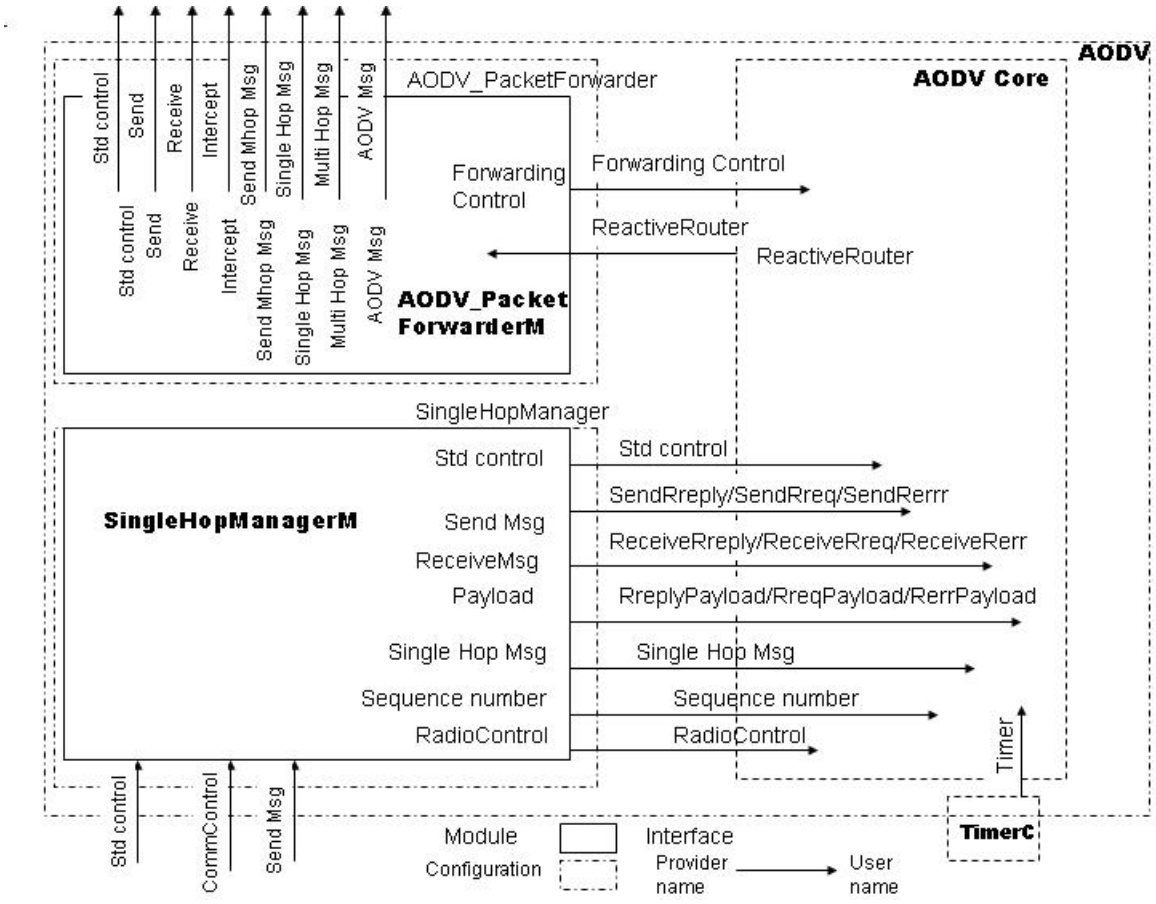


Figure 2.2: TinyOS interfaces for AODV.

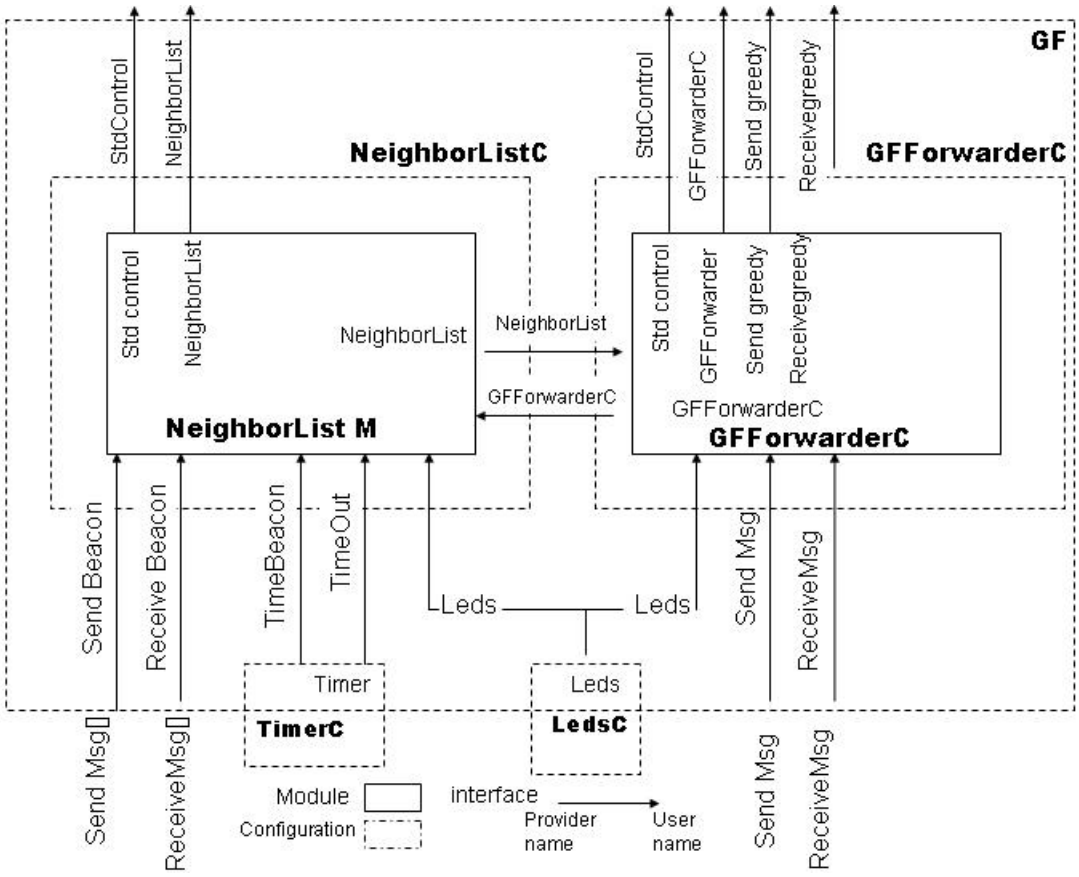


Figure 2.3: TinyOS interfaces for GF and GF-RSSI.

There are two implemented protocols that are based on geographic routing. One of them is GF (Greedy Forwarding) [9]. In greedy forwarding, the packet is transmitted to the neighbor of the sender that is closest to the destination. The other geographic routing algorithm is GF-RSSI (Greedy Forwarding with Received Signal Strength indication). GF-RSSI uses signal strength as one of the link estimator. If the sender finds a neighbor node closest to the destination and the signal strength from the neighbor is above a certain threshold, then it will forward the data to that node. Otherwise, the sender will search for another neighbor, which has a better link quality indication [8].

GF performs poorer than GF-RSSI, because communication paths frequently become unreliable due to the interference by neighboring communications, therefore choosing the shortest path may not always be the best solution. Making use of RSSI metric helps solving this issue

2.2 RSSI Metric for Estimating Link Quality

When we observe the multihop routing protocols mentioned in the previous section, the most critical part in the design of them is estimating the link quality. Two most preferred approaches for estimating the link quality are:

- Observing the number of packets received, and the number of packets missed from the source nodes.
- Looking into the RSSI value for the packets received from the source nodes.

Our protocol makes use of the second approach for deciding about link quality and selecting the parent node (see Chapter 5).

RSSI can also be used to estimate the distance. However, using RSSI as an estimation for distance has some challenges. In indoor environments, received signals fluctuate a lot due to the multipath effect.

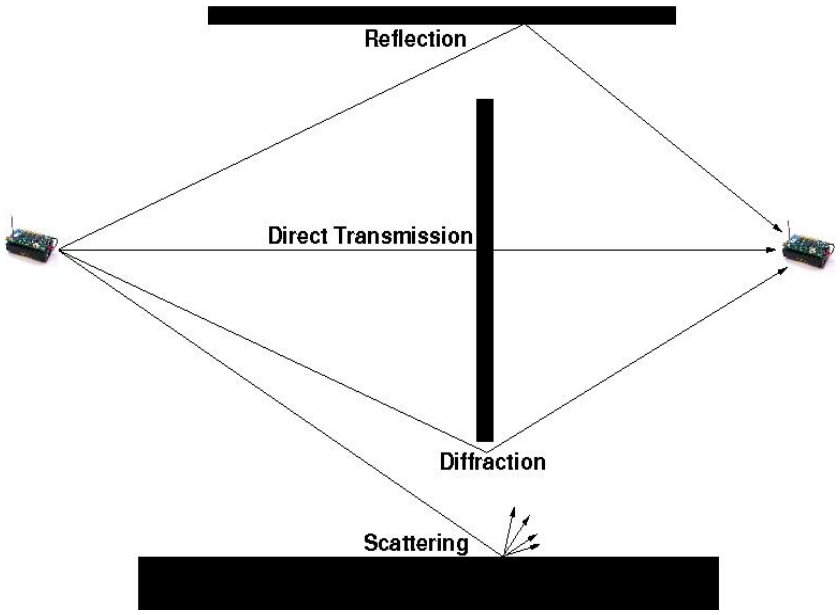


Figure 2.4: Propagation mechanisms that cause multipath effect.

Multipath effect [10] occurs when multiple versions of the same signal arrive to the receiver. A small movement of the transmitter or the receiver, and the objects around may fluctuate the received signal. Propogation mechanisms that cause multipath effect are shown in Figure 2.4.

Referring to Figure 2.4, it is obvious that indoor environment are prone to multipath effect, because reflection, diffraction and scattering are likely to happen in indoor environments, since indoor environments mostly include many obstacles. In our case, the office environment has some cubicles which cause all types of multipath effect during transmissions.

There are various papers in the literature offering solutions to inaccurate measurements of RSSI values in indoor environments.

One of the papers is [11]. This paper states that there is a limited relation between RSSI value and the real distance when the measurements are done indoor environment. In addition, there is a set of influencing factors, which can be controlled and exploited for improved measurement results. These factors are:

- Antenna characteristics and orientation,
- Variation of the transmit power,
- Variation of the frequency.

The transmit power and the frequency determine the maximum range of the radio waves. While the maximum transmit power might be appropriate for long distance communication (disregarding energy requirements), differences in the RSSI are hardly visible for small distances between transmitters and receivers. However, the measurement of short distances for the localization in closed areas with small dimensions is important. Thus, the transmit power must be well-controlled for meaningful RSSI based distance requirements. Considering this fact, we use default transmit power, while transmitting control messages in our E-Sense system.

Normally, RSSI values are inversely proportional with distance. In their paper they measured the values using CC1000 radio [11]. In the specifications of CC1000 radio, the directly measured value (to which we refer as ADC value) can be converted to RSSI value using a formula. Instead of converting the ADC value to the RSSI value, they chose to use the ADC value itself. The ADC value is directly proportional with distance. Consequently in the measurement results of this paper, when the distance increases, the numerical results of measurements (ADC values) increase as well.

In our E-Sense system we also used CC1000 radios, but we mapped ADC values to RSSI values in a different manner.(see Chapter 5)

In Figure 2.5, it can be clearly seen that the measured values in indoors vary strongly under identical conditions. The constant distance was 2.2 m. There were only 9 different values measured from the range of ADC values between 15 and 45. The measurements cover limited range of values, some optimizations could be done to convert these inaccurate data to accurate [11].

For the distance estimation, it is crucial that a small number of measurements (Figure 2.6) are enough for supplying relevant results. The comparison of both

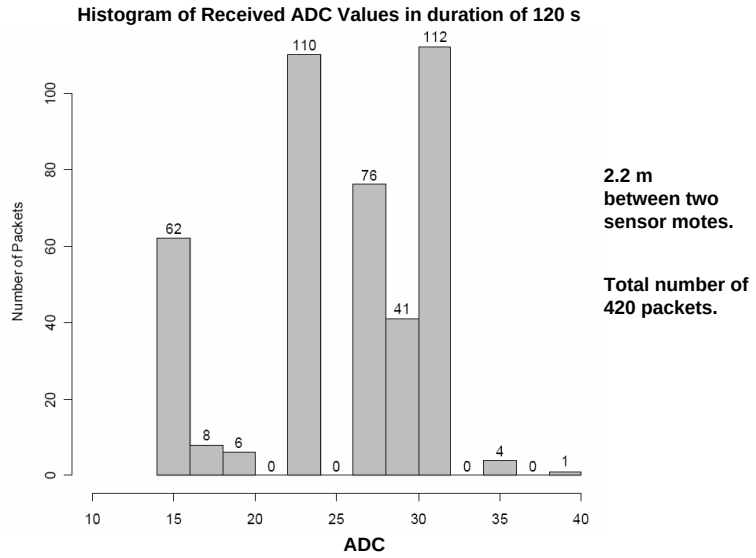


Figure 2.5: Variation of the ADC values over 120s.

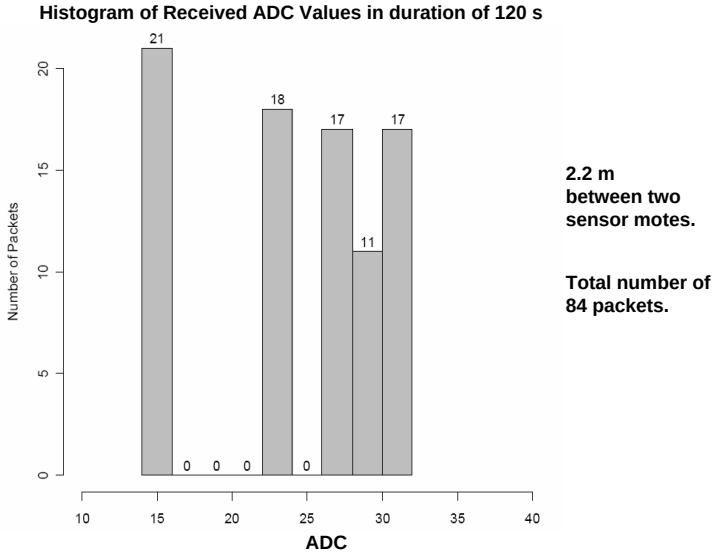


Figure 2.6: Variation of the ADC values over 20s.

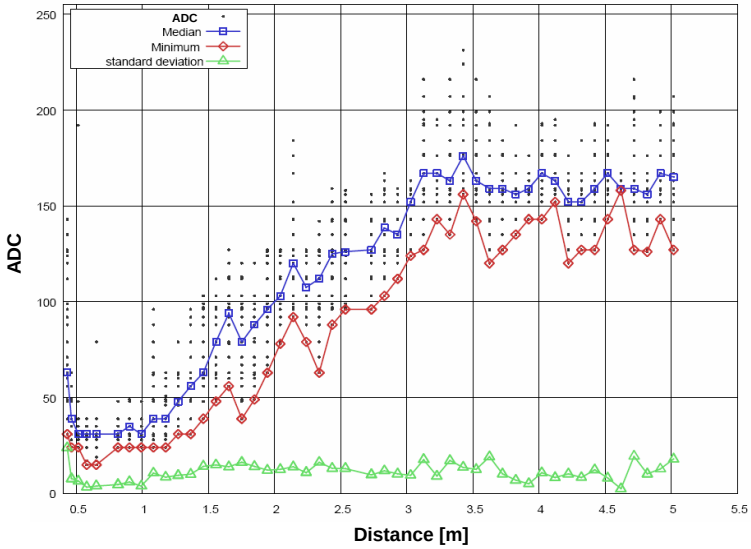


Figure 2.7: Linear and exponential regression based on the median of the measured ADC values. ADC measurements were done using 10 cm step size.

histograms (Figure 2.5 and 2.6) shows differences in the distribution of the RSSI values. Nevertheless, the statistical properties need to be considered. One of the statistical properties is that the mean values of measurements in Figure 2.5 and 2.6 are 25.75 and 26.75 respectively. A comparison of the statistical data shows that both measurements exhibit similar characteristics. [11].

The authors of this paper also observed the smoothness of ADC measurements. They measured ADC values starting from the distance 0.5 m to 5 m, incrementing the distance 10 cm in each step. According to Figure 2.7, in the ranges between 0.5 m and 1.0 m and also between 3.5 m and 5.0 m, the ADC values hardly contain information about the distance. The measured values oscillate without a tendency. However, between 1.0m and 3.5m, the measured values seem to be meaningful.

There are some suggested solutions for more accurate ADC values. In Figure 2.7, we can see the linear and exponential regression methods applied to ADC

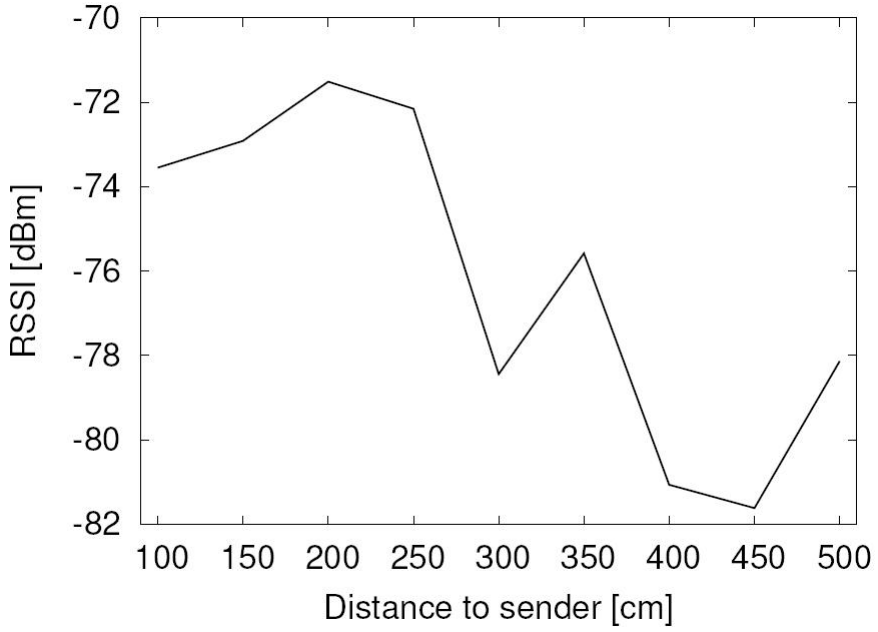


Figure 2.8: RSSI vs Distance in indoors.

values. It seems that using median method estimates the distance better than other methods. Our E-Sense system also uses CC1000 radio, and we chose taking average of measured values. We believe this solution is simple and effective.

In the M.Sc. thesis called "Fine-grained Indoor Localisation using Wireless Sensor Networks" [12], the indoor-measurements of RSSI is taken and plotted against distance, which is shown in Figure 2.8. This plot reflects the behaviour of RSSI better in indoors, because extensive number of measurements was taken at each point.

In another paper, "Wireless Sensor Networks and Radio Localization: a Metrological Analysis of the MICA2 received signal strength indicator" [13], the authors observed that as the battery power of the receiver decreases, the RSSI value measured on the receiver becomes lower as well. The outcome of this fact is, if we construct the route based on the received signal strength values, we will also establish a path so that the less energy remained in the battery, the less probable will the node be used as a relay node. Consequently, the consumed energy in the

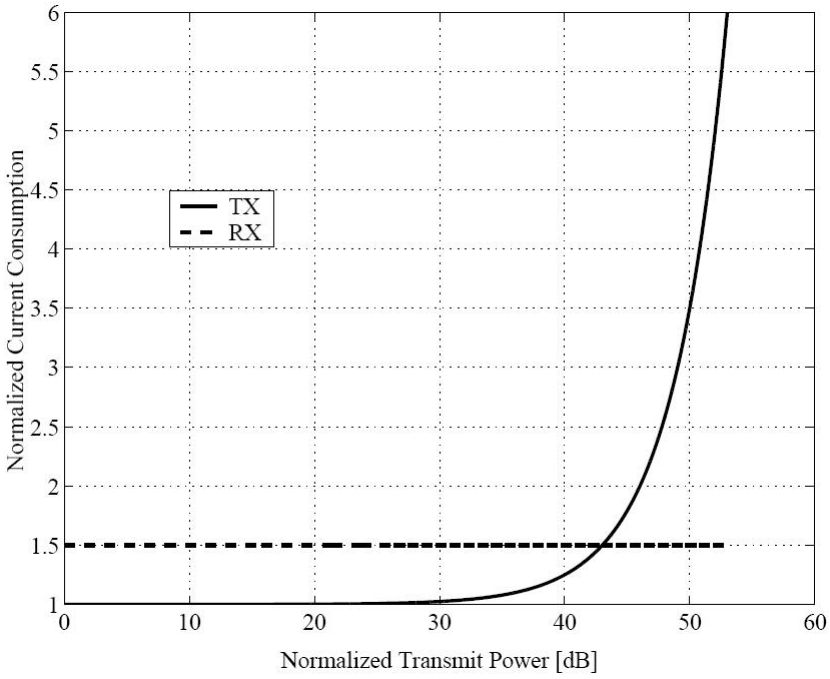


Figure 2.9: Energy Consumption vs Transmit Power.

nodes are evenly distributed throughout the sensor network.

2.3 Transmit Power Evaluation

In our E-Sense system, sensors are capable of adjusting their transmit power. Our sensors use CC1000 radio (see Section 5) operating at 915 Mhz. There are 26 different transmit power levels which could be assigned at both the compile-time and the run-time [14].

Our multihop routing algorithm aims at preserving more energy by decreasing and increasing the transmit power. Figure 2.9 [15] shows the energy spent with respect to transmit power. After 40 dB, energy consumption increases exponentially as the transmit power increases. In other words, decreasing transmit power without affecting the overall topology saves us a lot of energy for each individual node. Energy spent while receiving the data is not affected by transmit power.

There is a strong relation between the transmit power and the RSSI values. Increasing and decreasing transmit power at the source node changes the RSSI values at the destination [16]. This may result in incorrect estimation of the distance based on the RSSI values.

The concepts about the previous multihop routing algorithms, the RSSI and the transmit power evaluation is covered in this chapter. An effective usage of the combination of these concepts leads us to construct an energy-efficient algorithm.

Chapter 3

Sensor Platform Information

Technological advances in electronics led scientists to develop small and low-powered sensors. These sensors can be capable of communicating through the wireless channel. These sensors are also reprogrammable, so that the implementation and the execution of complex programs like multihop routing and data reporting are possible.

This chapter covers the hardware and software components of the sensor platform we used in our E-Sense system.

3.1 Hardware

3.1.1 Mica2 (MPR400)

The MICA2 Mote is a third generation mote module used for enabling low-power wireless sensor networks. MICA2 is produced by Crossbow Inc. MICA2 has 868/915 MHz multi-channel transceiver which can operate with extended range. MICA2 is especially developed to be used for ad hoc wireless sensor networking. There are wide range of sensor boards and data acquisition add-on boards compatible with MICA2. MICA2 works with TinyOS, an open-source

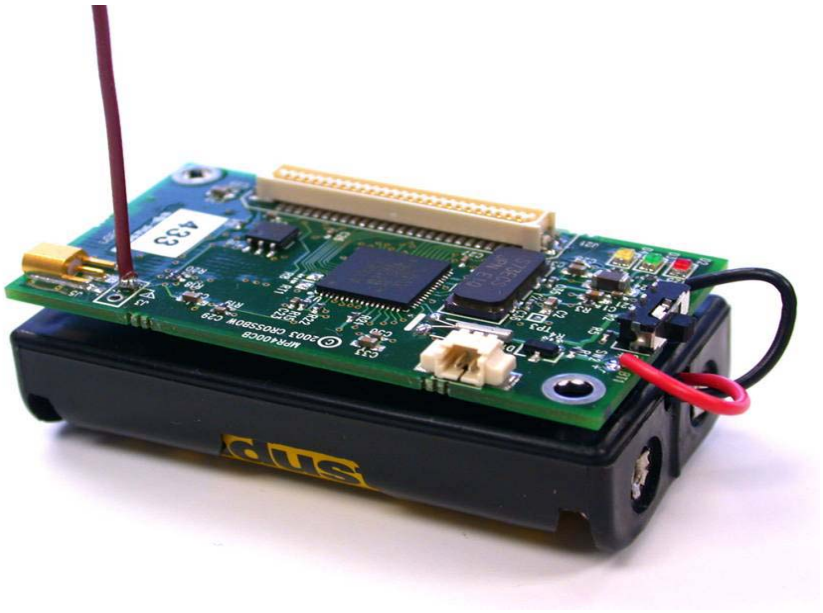


Figure 3.1: Mica2 processor board.

operating system for wireless sensor networks [17].

MICA2 is based on the Atmel ATmega128L chip. The ATmega128L is a low-power microcontroller which runs MoteWorks from its internal flash memory. A single processor board can be configured to run the sensor application/processing and the network/radio communications stack simultaneously. The MICA2 51-pin expansion connector supports Analog Inputs, Digital I/O, I2C, SPI and UART interfaces. These interfaces make it easy to connect to a wide variety of external peripherals. MICA2 motes work with 2 AA batteries [17].

3.1.2 Mica Sensor Board (MTS300)

Mica sensor board includes a light and temperature sensor, and also a component with a microphone and sounder in it. This board is designed for Mica2 processor board which can be connected via the 51-pin expansion connector. Its sound component can be used for acoustic ranging as well as measuring the noise in the environment [18].

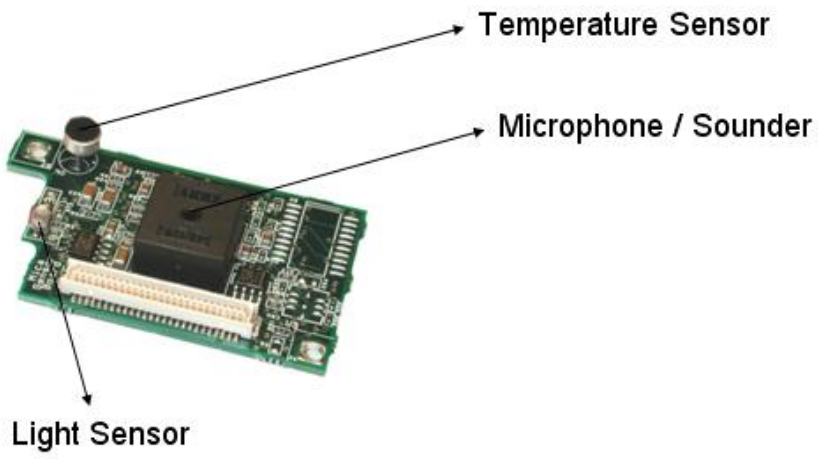


Figure 3.2: Mica sensor board.

3.1.3 Serial PC Interface Board (MIB510)

The MIB510 interface board allows the aggregation and collection of sensor network data on a PC as well as other standard computer platforms. Any MICA2 node can function as a base station when mated to the MIB510 serial interface board. In addition to the data transfer, the MIB510 also provides an RS-232 serial programming interface. The MIB510 has an onboard processor that programs the Mote processor/radio boards. The processor also monitors the MIB510 power voltage and disables programming if the voltage is not within the required limits. Two 51-pin-connectors are available, allowing sensor boards to be attached for monitoring or code development [19].

3.2 Software

3.2.1 Operating System

The operating system used in sensor nodes is TinyOS. TinyOS is an open source, flexible operating system built from a set of reusable components that

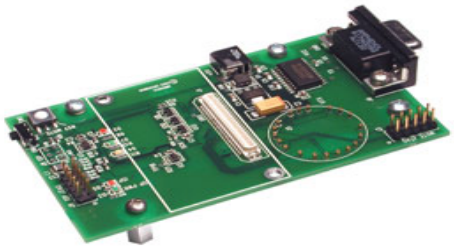


Figure 3.3: Serial PC interface board.

are assembled into an application-specific system. TinyOS supports an event-driven concurrency model based on split-phase interfaces, asynchronous events, and deferred computational elements called tasks. TinyOS is implemented in the NesC language [20]. NesC supports the TinyOS component and concurrency model.

A TinyOS program is a graph of components, each of which is an independent computational entity that exposes one or more *interfaces*. Components have three computational abstractions: *commands*, *events*, and *tasks*. Commands and events are mechanisms for inter-component communication, while tasks are used to express intra-component concurrency.

A command is typically a request to a component to perform some service, such as initiating a sensor reading, while an event signals the completion of that service. Events may also be signaled asynchronously, for example, due to hardware interrupts or message arrival. Rather than performing a computation immediately, commands and event handlers may post a task, a function executed by the TinyOS scheduler at a later time. This allows commands and events to be responsive, returning immediately while deferring extensive computation to tasks. While tasks may perform significant computation, their basic execution model is run-to-completion, rather than to run indefinitely; this allows tasks to be much lighter-weight than threads. Tasks represent internal concurrency within a component and may only access state within that component. The standard TinyOS task scheduler uses a non-preemptive, FIFO scheduling policy [21].

There is a simulation environment called TOSSIM in TinyOS. Instead of compiling a TinyOS application for a mote, users can compile it into the TOSSIM framework, which runs on a PC. This allows users to debug, test, and analyze algorithms in a controlled and repeatable environment [22]. We got help from TOSSIM when debugging and testing our code.

There are currently two different versions of TinyOS in the community, version 1.x and 2.x. In our E-Sense system, the version 1.x is used, because it is used for many years and more complete. The second version 2.x is continuously evolving. It is stated by the developers that TinyOS 2.x is platform-independent and more robust, but it currently lacks many things such as a multihop routing architecture, which we used in our E-Sense system. TinyOS 1.x and 2.x does not have source-code compatibility, making it challenging to convert the 1.x version code to 2.x version.

In a wireless sensor network using TinyOS, each node is assigned an ID number in the network, an analogy to the IP addresses in the Internet.

3.2.2 MAC Protocol

MAC protocols are implemented in software as the components of TinyOS. The default MAC protocol adopted by TinyOS developers is B-MAC [23].

Features of B-MAC are:

- Low Power Operation,
- Effective Collision Avoidance,
- Simple Implementation, small code and RAM size,
- Efficient Channel Utilization at Low and High Data Rates,
- Reconfigurable by Network Protocols,
- Tolerant to Changing RF/Networking Conditions,

- Scalable to Large Numbers of Nodes.

Reconfigurable means that we could change the MAC parameters, like the duty cycle percentages of the receiver and the transmitter, easily without going through the details of B-MAC implementation.

Application developers are given the option to use other MAC protocols as well, like S-MAC [24] and T-MAC [25].

Chapter 4

Our E-Sense System

Wireless sensor networks are very suitable for monitoring the environment. The monitoring may include measuring the brightness of the light, the environment noise, and the temperature. In this thesis, we introduce E-Sense, a wireless sensor network environment monitoring system with a user-friendly web interface for submitting queries and getting feedbacks.

The architecture of the E-Sense system is shown in Figure 4.1. The system is divided into layers so that each layer interacts directly only with the layer immediately beneath or above it. The components of the system are:

- **User Layer:** Users can submit queries and see the immediate feedback of these queries. Users can also observe the query-independent measurements done only for monitoring purposes.
- **Intermediate Layer:** This layer manages forwarding the queries to the sensor network and the results to the user.
- **Sensing Layer:** This layer consists of many sensor nodes and a sink node. They are responsible for getting and forwarding the measurements to the Intermediate Layer.
- **Database Server:** Database server stores all the past measurements.

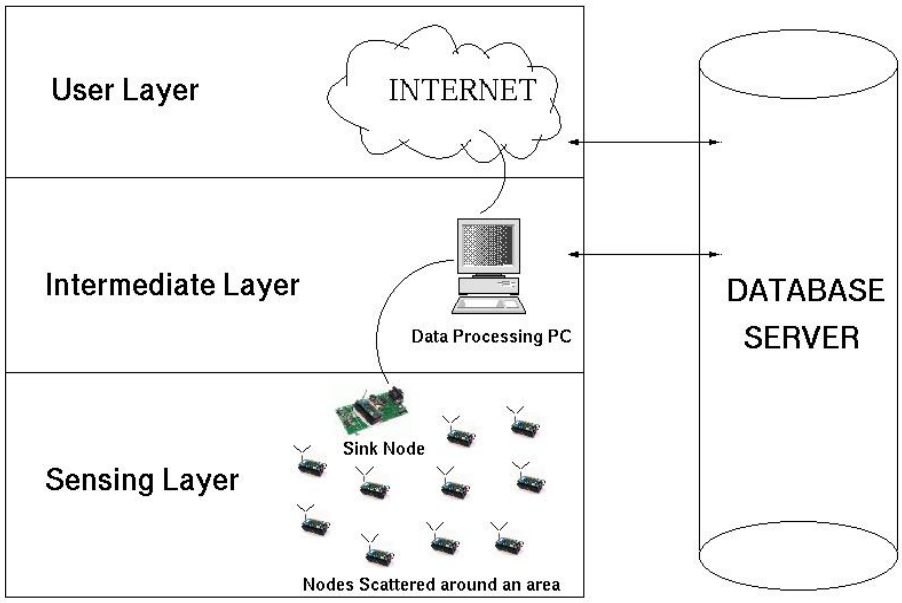


Figure 4.1: General structure of the E-Sense system.

These measurements can be queried by a user at any time.

These compenents together construct the system. Each of the layers runs variety of processes. Figure 4.2 shows the detailed architecture of the system. It illustrates the system flow from the clients to the wireless sensor network along with responsibilities of each process.

4.1 User Layer

This layer directly interacts with a user. The user can send queries and obtain the corresponding results via this layer. This layer simply consists of several web pages/scripts which run on an Apache web server. These scripts are implemented in PHP. When the user enters the URL `http://nbberker.cs.bilkent.edu.tr` of the web-site to his/her web browser, he/she is welcomed with a menu-like homepage (see Figure 3). The user is prompted with 3 options:

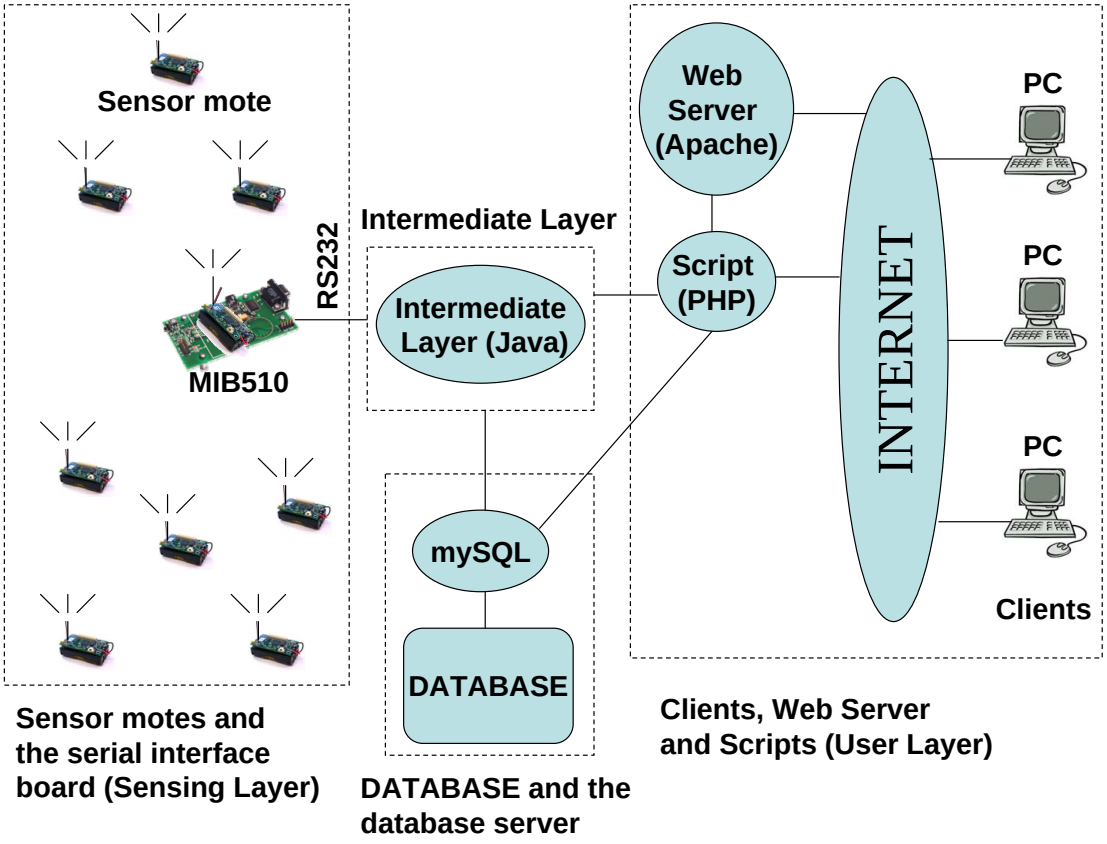


Figure 4.2: The architecture and components of the E-Sense system.

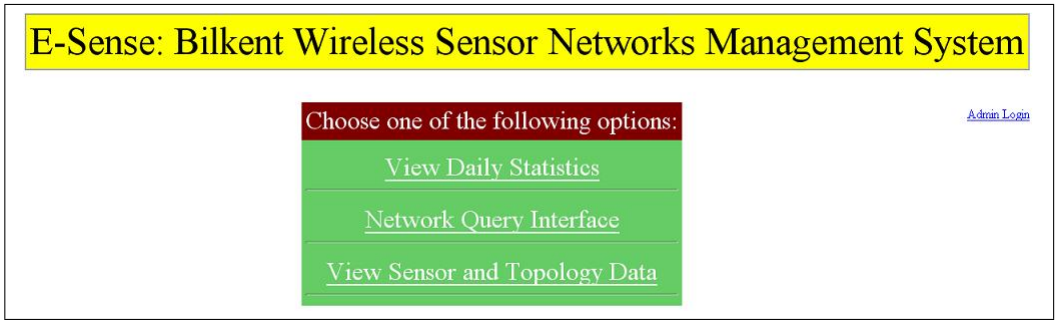


Figure 4.3: Entry page of the E-Sense system.

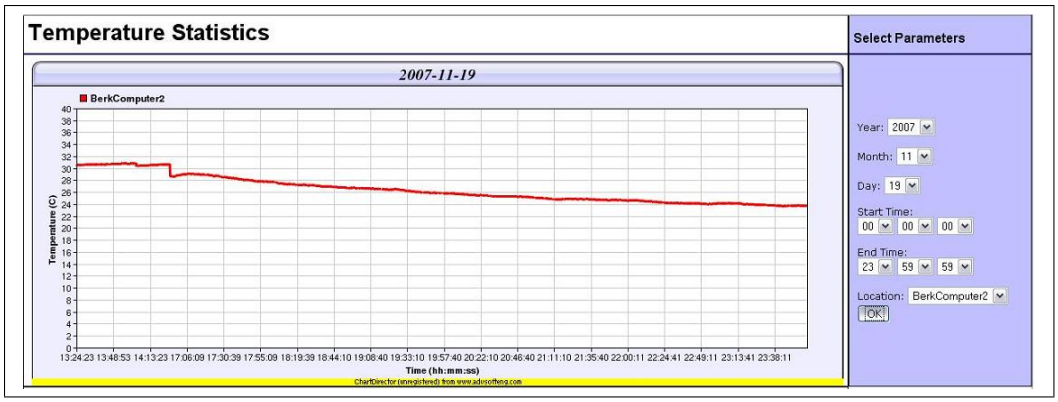


Figure 4.4: Temperature statistics along with the parameter selector.

- View Daily Statistics: Displays the measurements in an x-y graph format for a chosen day/month/year.
- Network Query Interface: In this page, the user enters a desired query by using the GUI components in the web page.
- View Sensor and Topology Data: This page includes a Java applet. This applet is divided into 2 panels. The first panel displays the incoming data from the sensors in textual format. The second panel displays the overall topology of the current sensor network.

View Daily Statistics: For each of the measurements, a corresponding x-y graph is displayed on the left side of the page. On the right side, the user can select a specific date&time for the measurements as well as the specific sensor node located at the given location. By default, the date&time displays the current date&time, and the start time is *00:00*, the end time is *23:59*, and the sensor location is the first entry gathered from the *Mapping* table of the database. There are 4 types of statistics to display: temperature levels, noise levels, light levels, and remaining battery energy levels. The statistical data are fetched from the database server.

Network Query Interface: The user is welcomed with a combo-box pre-selected a default option called "Choose query to send". Clicking the combo-box

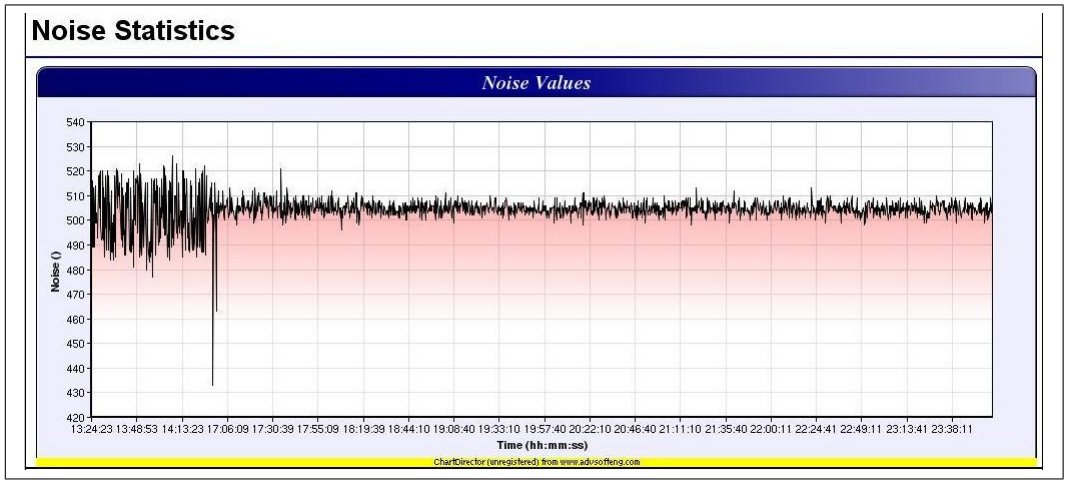


Figure 4.5: Noise statistics.

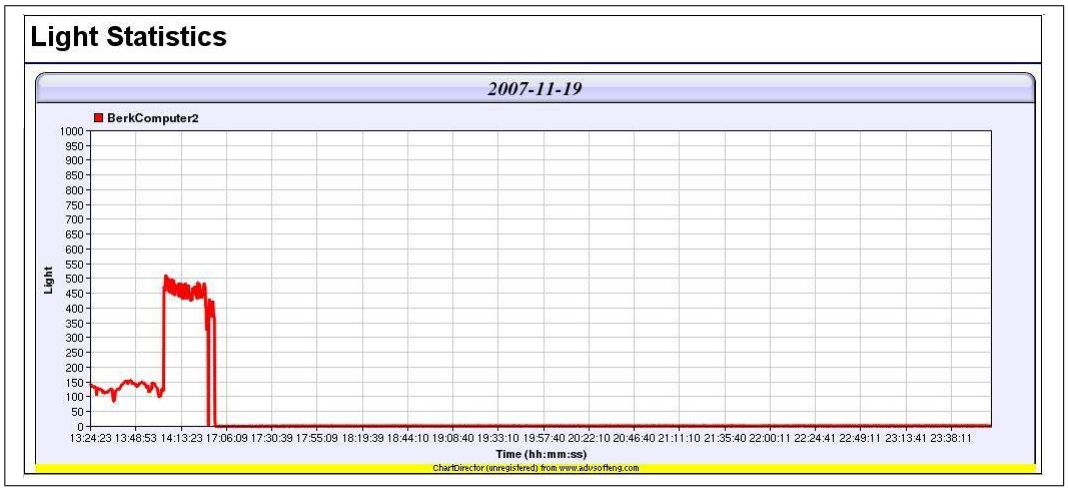


Figure 4.6: Light statistics.

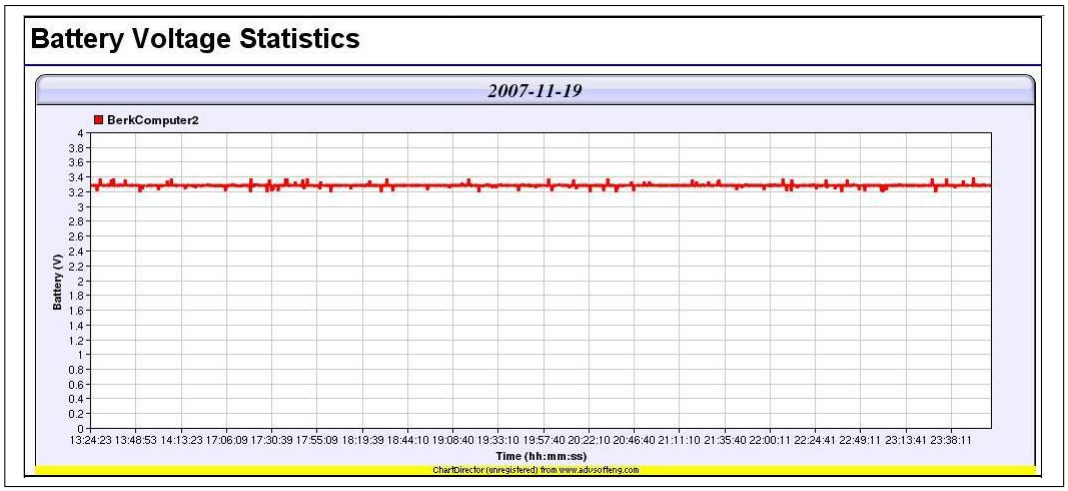


Figure 4.7: Battery statistics.

brings the types of queries for selection. There are 2 options to select, which are "Sampling Query" and "Search Results".

Sampling Query: In this type of query, the user enters the total duration of the data sampling, the interval of sampling, and chooses to send the query to a specific node or all nodes. Query submission is done by opening a socket connection to the Intermediate Layer and sending the query data to that layer. After a successful query submission, the sensor node sends the sensed-data to the sink node at specified intervals. The results can be displayed in "View Sensor and Topology Messages" page. The responses of the query submission go only to the corresponding owner of the query. The owner is determined by the session ID of the web-browser.

In the E-Sense system, we allow maximum two queries to be processed in parallel due to the limitations of the sensor motes. If the 3rd user tries to submit any query, he/she is informed with a message saying that there is not enough query slot available.

Search Results: In this page, we provide a GUI for the user to query the previous measured values. Firstly, the user is prompted with the type of query result which can be selected by a combo box. Depending on the user's selection,

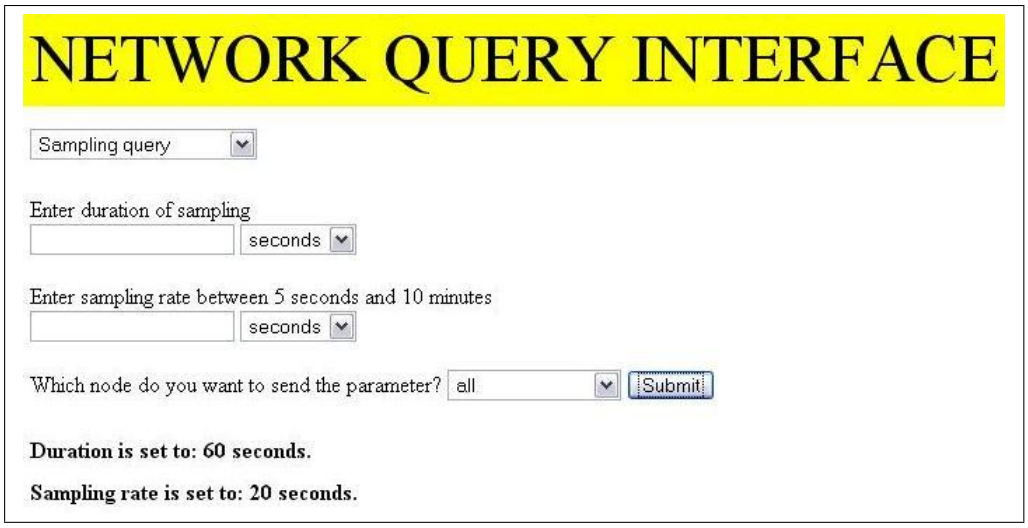


Figure 4.8: Sampling query interface.

the user is given the options to select the date/time interval, the interval of temperature/light/battery values, and the location of the originator node. The results are shown at the bottom of the web-page.

View Sensor and Topology Data: This page consists of:

1. On the left-hand side, an applet displaying the textual data and the topology view of the sensor network.
2. On the right-hand side, location mappings of the sensors.

In the first panel, the user can see the feedback of his/her issued query which is requested in "Network Query Interface" page. As soon as the data arrives to the intermediate layer, it is forwarded to this applet, so the textual feedback is displayed in a real-time fashion. The topology is displayed as a connected graph, consisting of vertices and edges. The connectivity of two sensor nodes can be broken by:

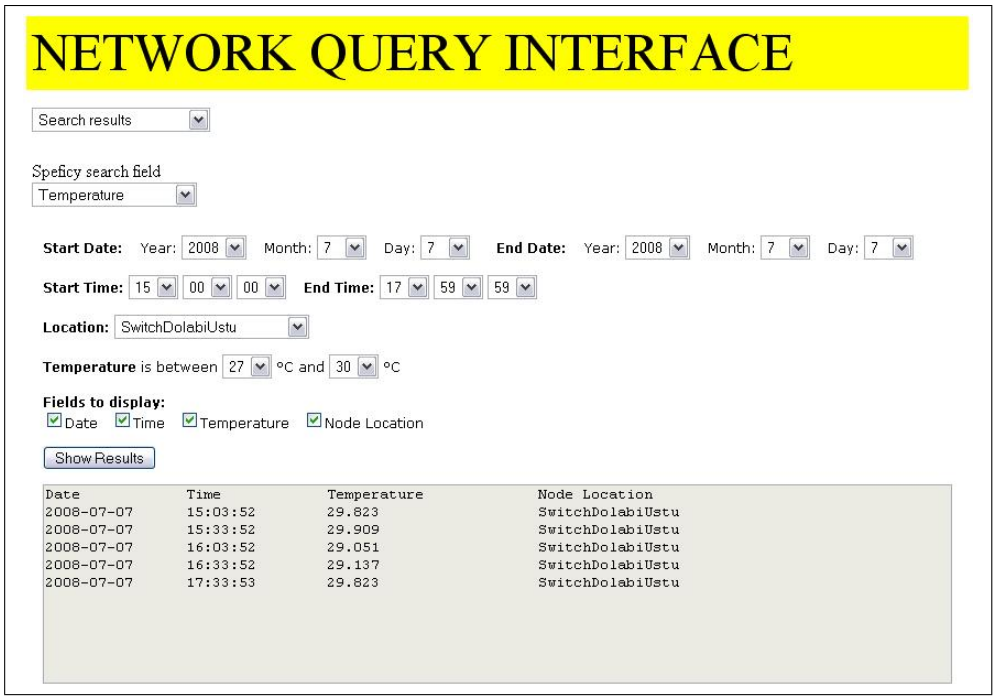


Figure 4.9: Search results page for temperature.

1. The mote is turned off or its battery dies.

2. The mote is moved further away until it is out of range.

Administrator Control: When a new node is to be added to the system, the administrator must add the *node ID* and the *location* of the node to the system. The administrator can also delete a node, or change the location of an existing node. The administrator page has to be restricted, so accessing to this page is done via a login system, requiring users to enter the correct username/password pair.

4.2 Intermediate Layer

This layer serves as a bridge, between the User Layer and Sensing Layer. Figure 4.14 shows the responsibilities of the Intermediate Layer. It forwards the

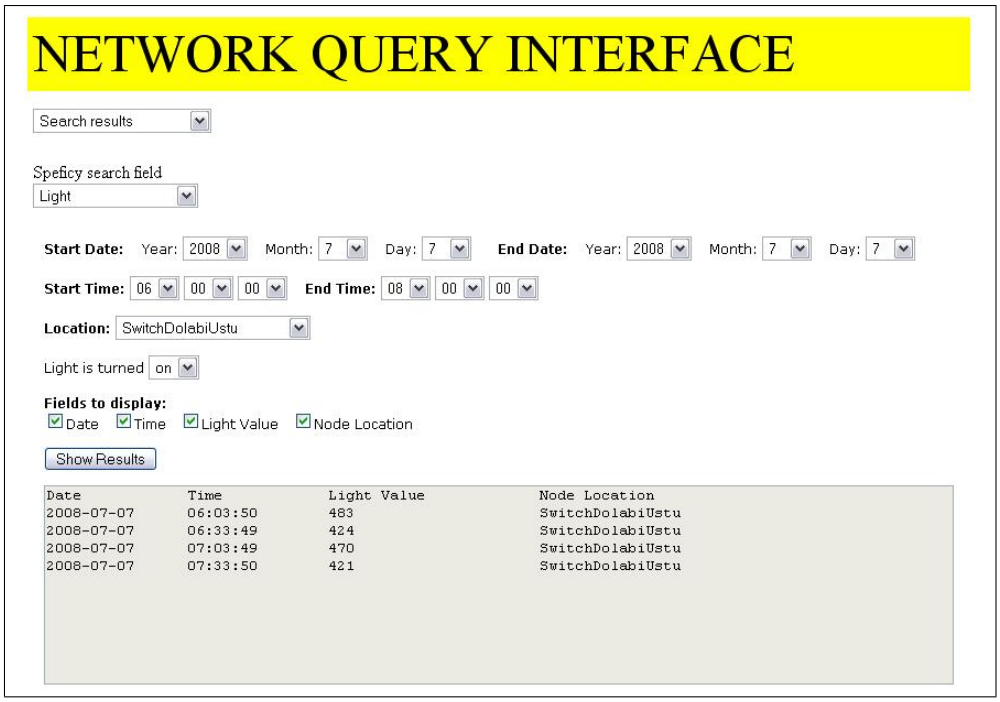


Figure 4.10: Search results page for light.

query submissions to the motes. It gets the query data of User Layer by listening a port (opening server socket handles it). It also gets the raw sensed data, converts the measurements to the scientific standards. The temperature data is converted according to the user manual of the sensor board [26]. Then, the meaningful sensor data is written to the database server, and it is forwarded to user-layer applets for displaying to the users.

We also obtain neighborhood messages from the Sensing Layer which help us to construct the topology. These messages arrive at some specific time intervals and are forwarded to the user-layer applets.

There are special java classes that stands as an interface between the sensor motes and the PC for the data sent and received. These java classes are generated through the "mig" (Message Interface Generator) of TinyOS. "mig" parses given header file and looks for structs that are also given as a parameter to "mig". The java classes are the representations of these message structs. The header files are

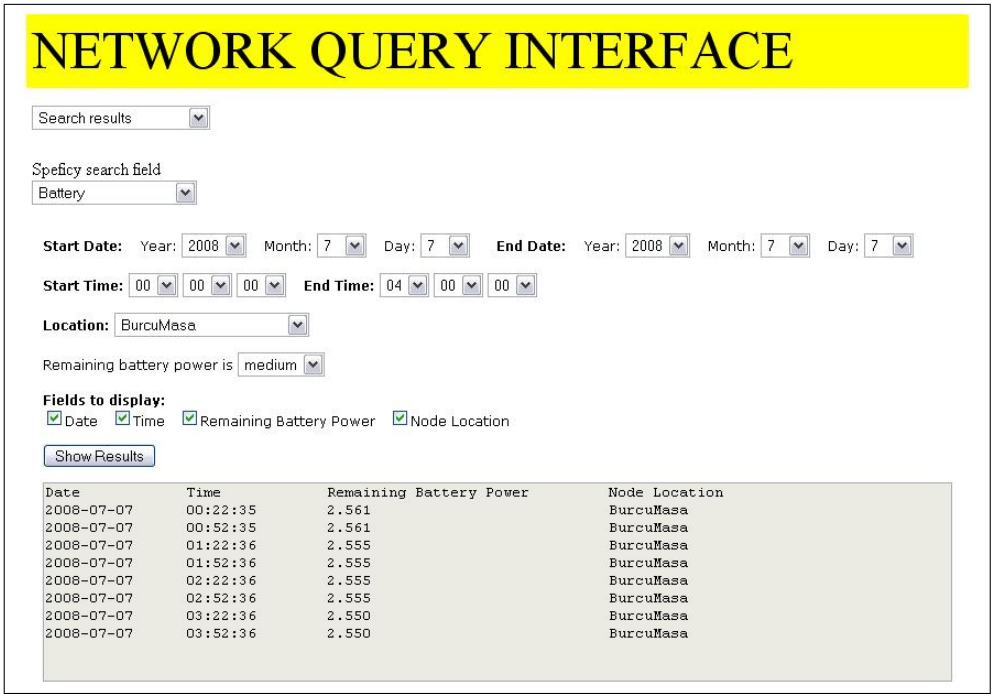


Figure 4.11: Search results page for battery.

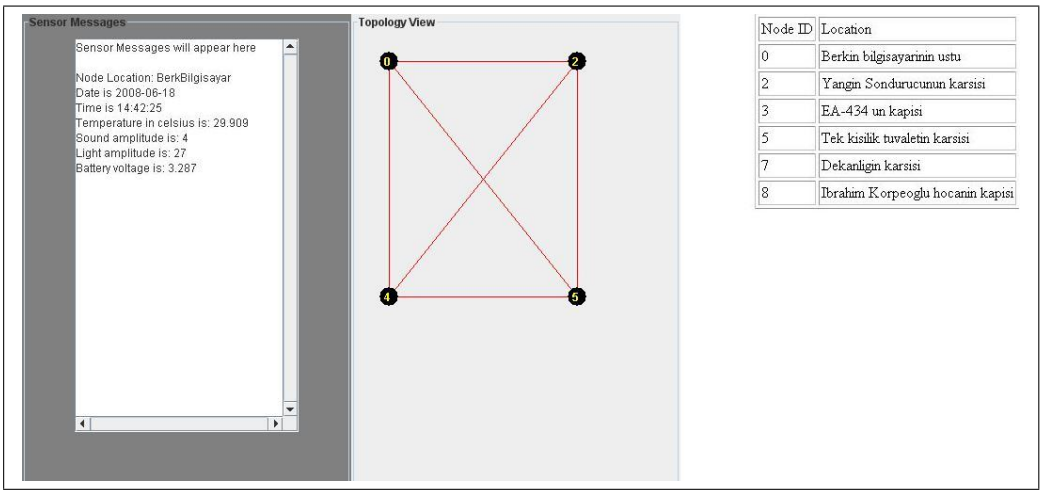


Figure 4.12: Arriving sensor data and topology view.

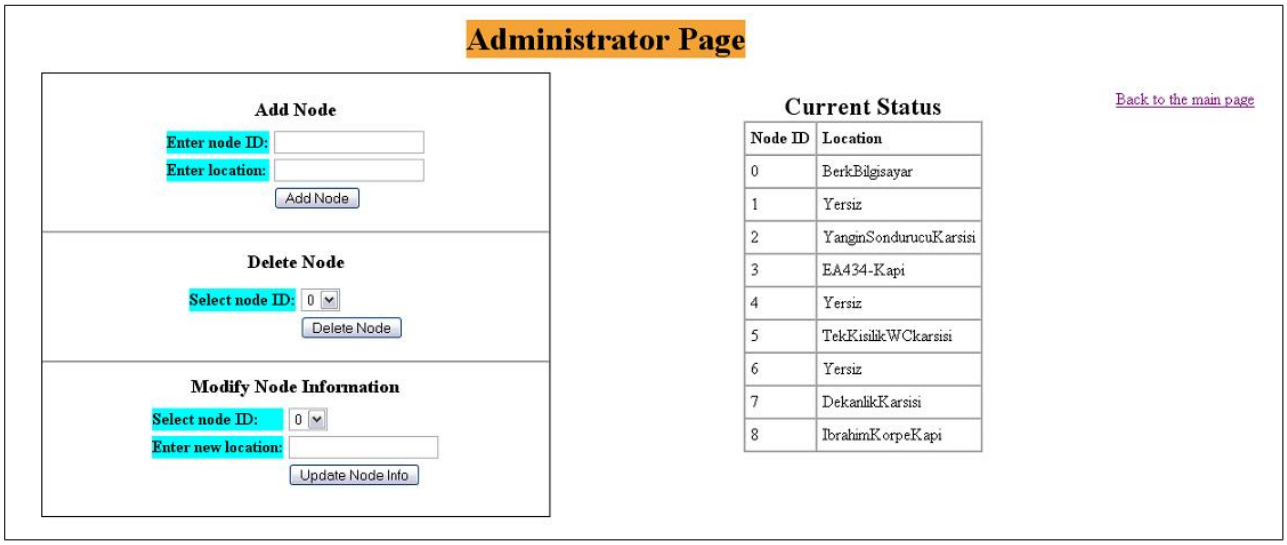


Figure 4.13: Administrator control page.

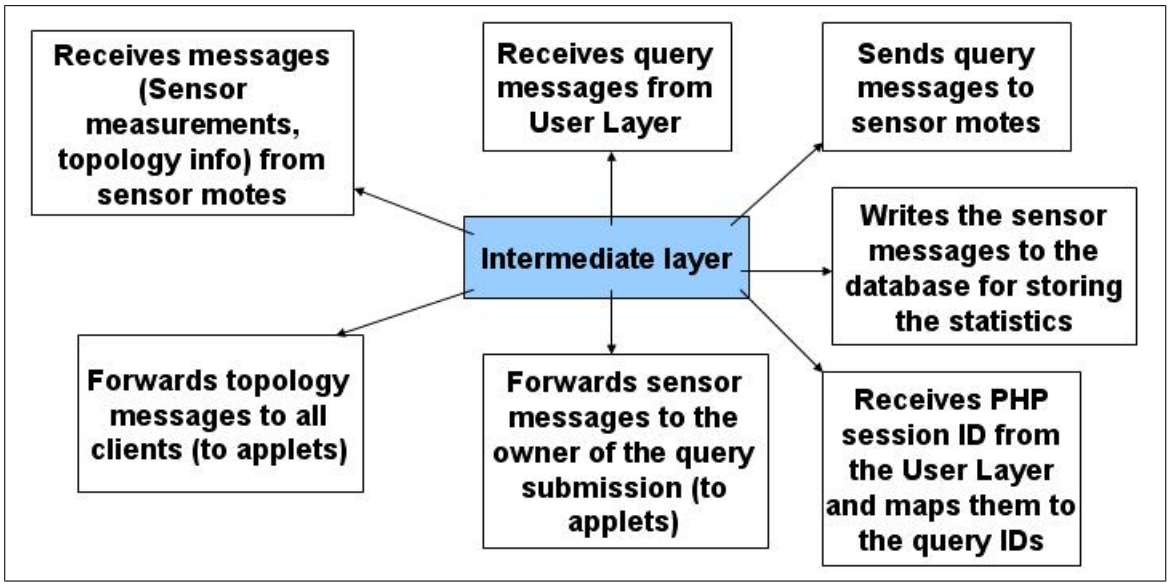


Figure 4.14: The jobs of intermediate layer.

Session ID	QueryID	Timestamp(sec)
9ab36b32cf432	1	40
7b629de5a2faa	2	30

Table 4.1: SessionID - QueryID mapping.

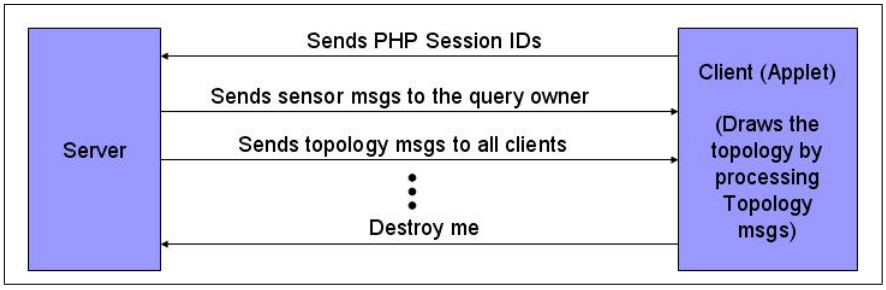


Figure 4.15: Message flow between the client applet and the intermediate layer server.

usually the part of the nesC program that are loaded to the motes. We will talk more about these structs in Section 4.3.

A PC is connected to the sink sensor mote (using the interface board) via the COM2 serial port, which is referred as the base station in our system. Queries are first sent to the base station and then disseminated from the base station to other motes. It is important to remind that the serial port is full duplex.

We support parallel query submissions. At the intermediate layer, queries are discriminated through the PHP session IDs. We store a list where each session ID has a corresponding query ID. At the sensing layer, queries are discriminated through the query IDs. Table 4.1 is an example of SessionID-QueryID mapping.

In User Layer section, we said that the replies of submitted queries only go to the submission owner, not to any other user. To achieve this, we must do some message exchange between the client applet and the server at the intermediate layer. Figure 4.15 shows this message exchange.

4.3 Sensing Layer

The Sensing Layer is the core of the system. This layer is responsible for getting the data from the environment, and sending the data to the base-station via the multi-hop routing algorithm we implemented.

The implementation of this layer is done on MICA2 sensor motes. The MICA2 platform is compatible with TinyOS, so we used nesC programming language of TinyOS for programming the motes.

Figure 4.16 shows an abstract diagram about application and network parts. Notice that network part handles all the network communication functions. HELLO (see Section 5.1.1) message is used to inform the neighbors about the parameters that help the algorithm decide the next node to send data.

The Sensing Layer is divided into 2 parts (both parts run in each sensor mote):

- Application Part: Application part is responsible for getting the measurements from ADC channels; receiving, processing the queries, and sending the sampled data to the base station. It uses network part for sending multihop data to the base station.
- Network Part: This part includes the multi-hop routing protocol implementation and provides multi-hop communication routines for application part.

4.3.1 Application Part

In this part, depending on the values in the query message, measurements are fetched and sent to the base station periodically. We make sure that we acquire all the measurements from ADC channels before sending them in a packet. To handle this situation, the program periodically (once in every 100 ms) checks whether all data from ADC are retrieved. If they are retrieved, the data is sent through the functions of the network part, which handles the multihop communication.

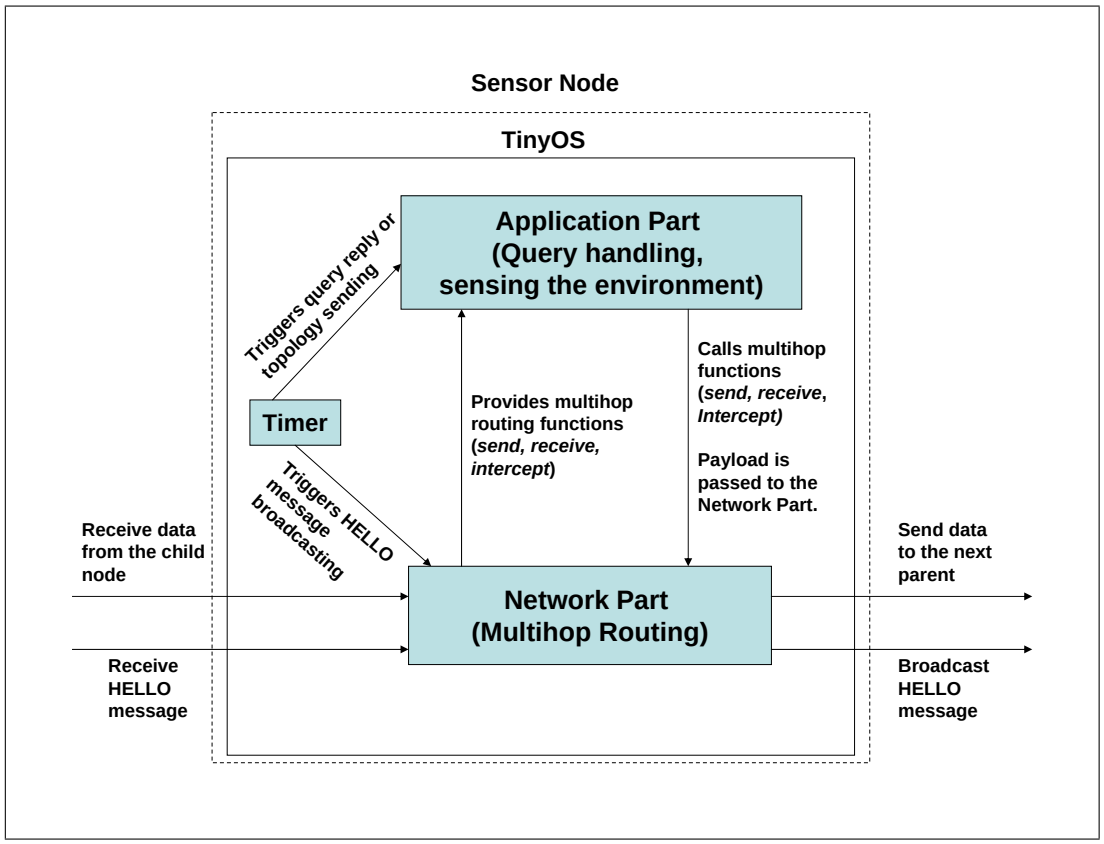


Figure 4.16: Block diagram depicting the relationship between the application part and the network part.

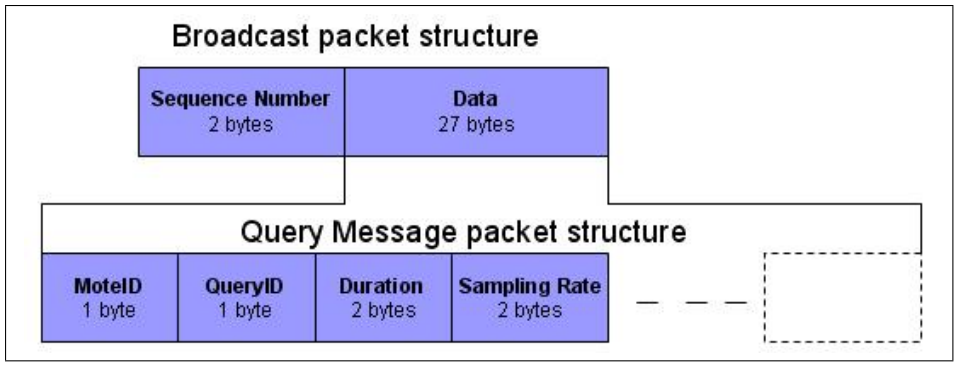


Figure 4.17: Query message format.

We also consider the case for parallel query submissions. In nesC language, dynamic binding of the memory is not allowed by TinyOS, because they claim that it would be very difficult for the developer to detect the possible memory leaks, so we limited the maximum number of queries that are processed at the same time. We allow at most two parallel queries to be processed.

Apart from processing queries on demand, the sensor motes independently monitor the environment and send the results once in every 30 minutes. We do this to observe the environment for very long times. "Daily statistics" and "search results" pages display these results, not user's query results.

The application part is also responsible for obtaining and sending the current neighbors of each node to the base station which is used to construct the network topology. The neighbor information is retrieved from the network part.

There are three types of messages that the application part handles. These messages are:

- **Query Messages:** When the user issues a query from the user layer, the intermediate layer generates a query message and sends this message to the sink node. The sink node disseminates the query message through all the sensor motes. Dissemination is done by flooding, i.e. each node broadcasts the message until it reaches every single node in the system. Each query

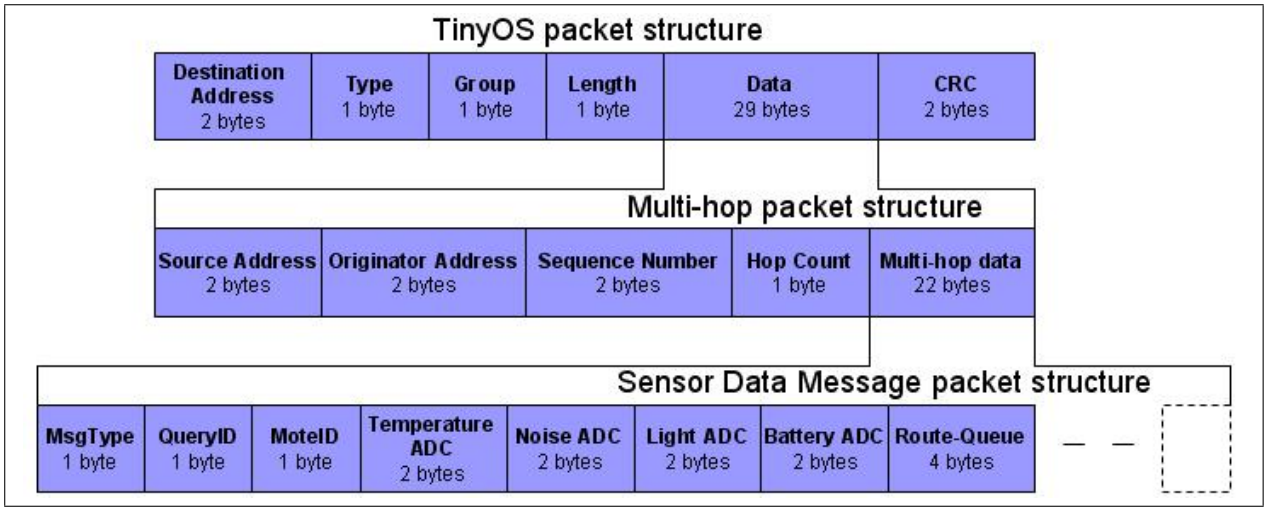


Figure 4.18: Sensor data message format.

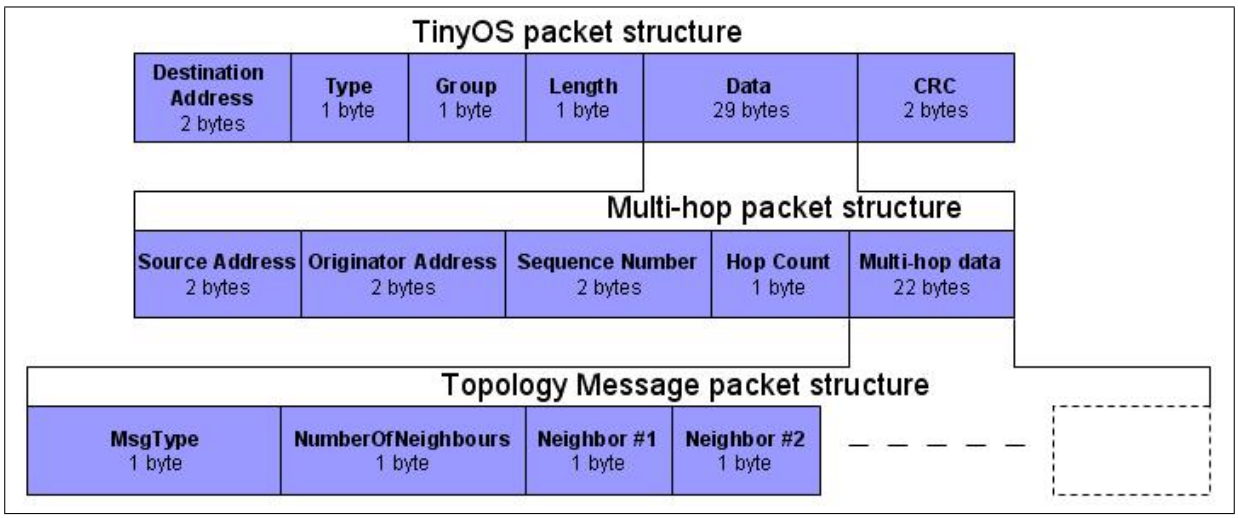


Figure 4.19: Topology message format.

messages initiates a timer, of which the period is retrieved from the query message itself. Each time the timer triggers, the sensor data message is generated and sent to the base station.

- **Sensor Data Messages:** A sensor data message keeps the measurements including the temperature, noise, light and the remaining battery energy (depends on voltage). A sensor data message is generated and sent if a query message triggers a timer or if the periodic monitoring timer is triggered (which is once in every 30 minutes).

- **Topology Messages:** Topology messages are later to be processed by the client applets for displaying topology of the sensor network. They are sent once in every 2 minutes by the sensor motes. Changes in the topology are reflected in the client applet quite rapidly.

Query message, sensor data message, and topology message formats are shown in Figures 4.17, 4.18, and 4,19 respectively.

The "MsgType" field in sensor data messages and topology messages is used by the intermediate layer to determine if the message includes the sensor data or topology data. The "RouteQueue" field helps us to learn the route that sensor data messages followed while reaching the base station. The sub-fields of RouteQueue are filled by using the active snooping function provided by the network part.

4.3.2 Network Part

Multihop routing and forwarding protocol implementation resides in this part. Detailed explanation of our multihop routing protocol is given in Chapter 5.

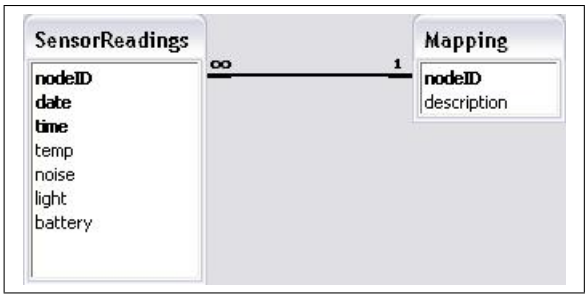


Figure 4.20: Database tables.

4.4 Database

We used mySQL server (version 5.024a) as the database server running on Linux.

Database structure is shown in Figure 4.20.

It consists of two tables. In SensorReading table, *nodeID* is the originator ID of the mote, *date* and *time* fields tell us when the measurements were taken, and the remaining fields contain the measurements from the motes. Primary key of *SensorReading* table is <*nodeID*, *date*, *time*> together.

We use *Mapping* table to store the location of each mote ID. Primary key of *Mapping* table is <*nodeID*>.

The *nodeID* field of *SensorReadings* table is also a foreign key to *nodeID* field of *Mapping* table. This is done to enforce the referential integrity, meaning that if an entry from *Mapping* table is deleted, all the entries of SensorReadings table which have the same *nodeID* with the deleted *Mapping* entry's *nodeID* are automatically deleted as well.

This chapter introduced the E-Sense system and its components. Each component contributes to the system's overall operation. The system serves the goal of the environment monitoring with a user-friendly web interface for submitting queries to the sensor nodes and getting feedbacks of physical mesaurements.

Chapter 5

Our Proposed Multihop Routing Algorithm

Sensor motes need to send their measured data to the sink node. To achieve this, a multihop routing algorithm must work in the sensor motes, so that they find routes to the base station. In our proposed algorithm, energy conservation is the most important issue that we considered. For extending the network lifetime as much as possible, we apply transmit power adjustment policy. When the parent node is near to the sender node, decreasing the transmit power increases the lifetime of each individual sensor node.

Our algorithm is a distance-vector routing algorithm with a many-to-one tree-based approach in which the root in the tree is the base station. The algorithm is executed once in every 60 seconds in a sensor mote, and after that the parameters of the node including the power cost to the sink node are broadcasted to neighbors of the node. In our algorithm, each node locally decides the parent node based on the minimized energy path to the sink node.

Initially, the power cost for reaching the sink node is not known by any node (infinity). After some time passes, the parameters span the whole network via the periodic HELLO (see Section 5.1.1) messages, and each node computes the total cost by adding the link cost to its neighbor with its neighbor's cost for reaching

the sink node. Among the neighbors, the one with minimum total cost is selected as the parent node. After selecting the parent, we adjust the transmit power. Transmit power adjustment requires some steps. First of all, each data packet produces RSSI value at the parent node. After the parent node broadcasts the average of these values, each child node retrieves its corresponding data packet's RSSI value. The transmit power adjustment is done based on this RSSI value. In order to apply this adjustment, the same parent must be selected both in the current execution and in the previous (60 seconds before) execution of the algorithm, because if the parent changes, we have to reset the power level to default level for ensuring the packet's delivery, since the new parent can be at any distance.

Figure 5.1 shows the TinyOS interfaces of our multihop algorithm. Part of the image was taken from [27]. In this figure, there are two modules that make the multihop communication work. The module *MultiHopBerkM* is responsible for informing and receiving the multihop parameters required for parent selection and transmit power adjustment, and making decision based on these parameters. Our algorithm runs in this module. The selected parent node ID is passed to the other module *MultiHopEngineM* through the *RouteSelect* interface. *MultiHopEngineM* module is responsible for sending and forwarding data packets to the parent node. *PowerLevel* interface is responsible for passing the transmit power value from *MultiHopBerkM* to *MultiHopEngineM*, so *MultiHopEngineM* module can change the transmit power before sending the data packet.

5.1 Implementation Details

5.1.1 Informing the neighboring nodes about the parameters

When a node is turned on, the node sends HELLO messages to inform its existence. A HELLO message is sent periodically, so that information about this node is kept up-to-date in neighboring nodes. The HELLO message is broadcasted by

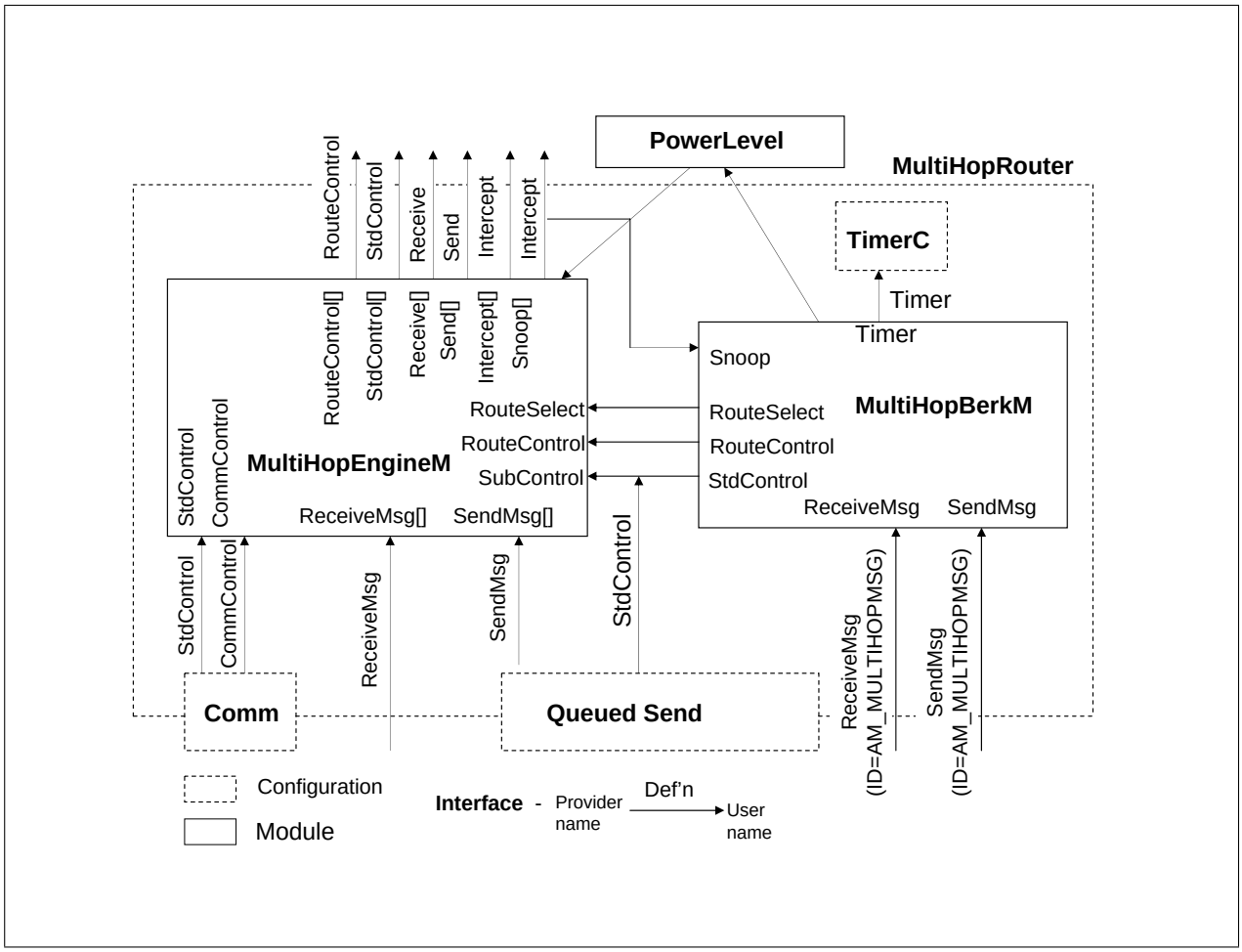


Figure 5.1: TinyOS interfaces of our algorithm.

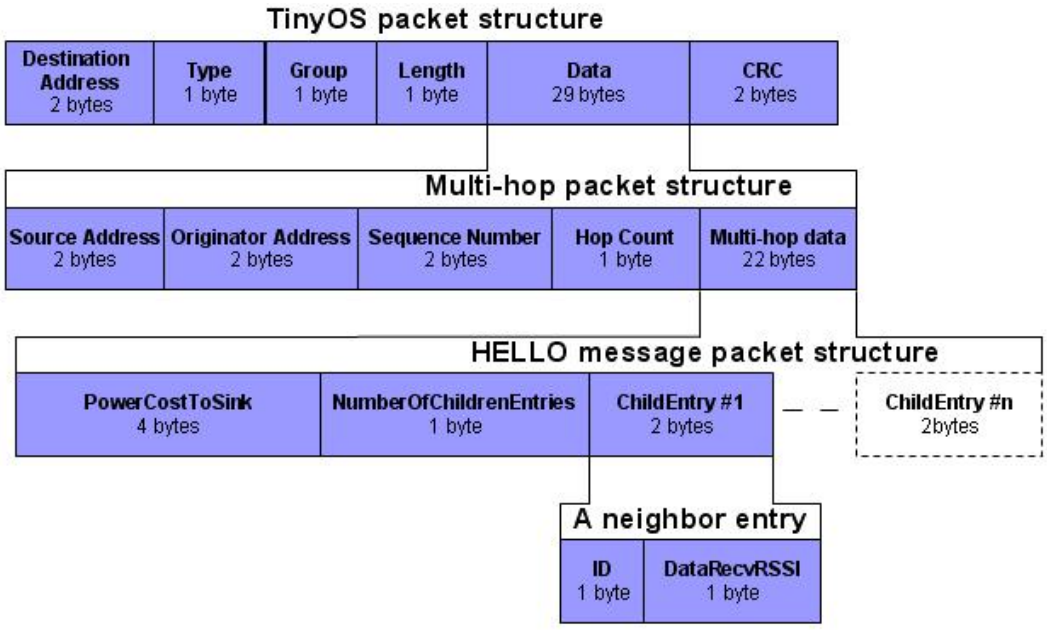


Figure 5.2: HELLO message format.

each node periodically, which is once in 60 seconds.

Figure 5.2 shows the fields of the HELLO message.

The fields of HELLO message are:

- **PowerCost:** Power cost from the neighbor node, which originates the message, to the sink node.
- **NumberOfChildrenEntries:** Stores the number of children entries in the packet.
- **ChildEntry-ID:** ID of the neighbor.
- **ChildEntry-DataRecvRSSI:** This field represents the RSSI value occured on current node when a child node sends a multihop data that comes from the application part. This value is sent to the child node in the next broadcast, which decides that the transmit power either increases or decreases.

Pout (dBm)	**PA_POW (hex) 915 MHz**	**Current Consumption (mA)**
-20	0x02	8.6
-19	0x02	8.8
-18	0x03	9.0
-17	0x03	9.0
-16	0x04	9.1
-15	0x05	9.3
-14	0x05	9.3
-13	0x06	9.5
-12	0x07	9.7
-11	0x08	9.9
-10	0x09	10.1
-9	0x0b	10.4
-8	0x0c	10.6
-7	0x0d	10.8
-6	0x0f	11.1
-5	0x40	13.8
-4	0x50	14.5
-3	0x50	14.5
-2	0x60	15.1
-1	0x70	15.8
0	0x80	16.8
1	0x90	17.2
2	0xb0	18.5
3	0xc0	19.2
4	0xf0	21.3
5	0xff	25.4

Table 5.1: Transmission power levels for CC1000 radio. Default power is 0 dBm.

5.1.2 Transmit Power Levels

Adjusting the transmit power is one of the capabilities of MICA2's radio which is called CC1000 radio. According to the manual of CC1000 radio [14], there are 26 different transmit power levels, and these values can be changed both in the compile-time and the run-time. We use this feature in our multihop routing algorithm to increase or decrease the transmit power. Table 5.1 shows the transmit power levels for 915 MHz frequency which we use in our sensor motes.

5.1.3 Calculating the RSSI value

In CC1000 radio, the signal strength of the incoming message is obtained from the *strength* field of TinyOS message structure. The value is directly proportional with the distance, because RSSI values given by CC1000 radio are not in dBm units. We referred this value as *ADC value* in Chapter 2.

The size of each ADC value is 2 bytes. We observed the behaviour of this value by increasing and decreasing the distance between two motes. On rare situations, the value exceeds the number 255, only when the distance between two motes is very high. Since the TinyOS packet has only 30 bytes of payload and we need to fill the fields related to RSSI in the HELLO messages, we decided to shrink the size of RSSI to 1 byte by scaling the value between 0 and 255 and subtracting the scaled value from 255. This means that the value 0 indicates no signal and the value 255 indicates maximum signal. Since RSSI values are periodically sent through the radio, we gain extra spaces in the packet payload for filling it with more information.

When mentioning the experiments in Chapter 6, the measured RSSI values indicate the scaled values between 0 and 255.

5.1.4 Duty cycle modes

As mentioned in Chapter 4, MAC parameters are reconfigurable from the application. The duty cycle mode is one of the parameters that can be changed through a set of functions. These functions are configured to change the idle listening ratio. The transmit mode function is used to set the preamble length and the receive mode function determines the check interval of the radio [23]. Duty cycles for both transmit and receive modes must be same for a reliable packet transmission. Table 5.2 [28] shows the duty cycle modes for the CC1000 radio. Default value for the mode in TinyOS is 0.

In our E-Sense system, the number of packets does not tend to exceed 4

Mode	**Duty Cycle** (%)	**Max Packet Rate (pkts/sec)**	**Effective Data Rate (kbps)**
0	100	42.93	12.364
1	35.5	19.69	5.671
2	11.5	8.64	2.488
3	7.53	6.03	1.737
4	5.61	4.64	1.336
5	2.22	1.94	0.559
6	1.00	0.89	0.258

Table 5.2: Duty cycle modes and their corresponding packet and data rates.

packets per second, except for very rare cases like multiple query submission at the same time. Considering this fact, the sensor motes are configured to run at duty cycle mode 4, which has the duty cycle percentage of 5.61%. Since sensor motes decrease their idle listening ratio, they last longer, extending the overall network lifetime significantly.

5.2 Routing Algorithm Details

The routing algorithm consists of two parts. In the first part, we select one of the neighboring nodes as a parent. In the parent selection process, we choose the one with the minimum power cost for reaching the base station. Second part handles adjusting the transmission power of the current node. The adjustment is done if the algorithm keeps selecting the same neighbor as the parent for each run.

5.2.1 Parent Selection

We use a distance-vector based routing protocol where each node locally selects one of the neighbors as its parent which has the minimum total cost for reaching the sink node.

Initially, each node has a power cost of infinity (∞) to reach the sink node except the sink node itself, which has the cost of zero (0). First of all, when one hop neighbors of the sink node receive the parameters, they simply add the link cost with the current cost to the sink (0). Furthermore, two hop neighbors receive the parameters and compute the cost, and this pattern goes on until the parameters span the whole network. In the final configuration, each node knows its parent and all routing paths are established.

RSSI values are used to estimate the power consumption costs for wireless links. Power consumption estimation using RSSI values is done as follows:

According to the first order radio model [29]: $E_T \ \alpha \ d^2$
where E_T is the energy spent and d is the distance between the transmitter and receiver.

The relation between RSSI and the distance is: RSSI α $(1/d^2)$
where RSSI stands for the signal strength value measured at the receiving node when the transmitting node sends a packet.

Thus, we come up with: $E_T \ \alpha \ (1/\text{RSSI})$.

This means that we can use the value (1/RSSI) for estimating the energy spent while transmitting through a wireless link.

RSSI values are extracted from the header of TinyOS messages (via HELLO messages) received from neighboring nodes, thus nodes learn RSSI values for each of their neighbors. We obtain more accurate RSSI values by taking average of last 10 measurements, resulting with a better power consumption estimation for wireless links.

When a message is received, the total cost is computed and stored in the neighbor table. Once in every 60 seconds, our algorithm determines the parent node which is the one with the minimum total cost.

The evaluation of the power (energy) consumption of links is done much like the Bellman-Ford algorithm [30]. The difference is that instead of storing the costs to all destinations, we only store the cost to the sink node. Using the terminology of Bellman-Ford algorithm, we refer to the total power cost for reaching the sink node as the *distance* of a node. Algorithm 1 gives the details of the parent selection algorithm.

Algorithm 1: The parent selection algorithm.

Naming conventions: *A-Xrssi* means the RSSI value measured on A when it is sent by X in a HELLO message to A (we assume the links are symmetric, this means the RSSI value measured on A are the same as the RSSI value measured on X). The total power cost from node X to the sink node is referred as *distance*.

Input: All neighbors. We are on node A.

Output: The parent node is selected. We are on node A.

```
minimumDistance = ∞;
parentNode = NIL;
foreach All Neighbors of A, which is X do
    nbrDistance = X.distance;
    if ( nbrDistance + (1/A-Xrssi) ) < minimumDistance then
        minimumDistance = nbrDistance + (1/A-Xrssi);
        parentNode = X;
    end
end
return parentNode;
```

The time and space requirements of the algorithm can be observed by using the complexity analysis. For the time complexity analysis, let G = (V, E) be the connectivity graph of the network topology, where V = {v_1, v_2, ... , v_n} in which v_i denotes the i^{th} node, and (v_i, v_j) $\in$ E if and only if v_i and v_j are reachable by each other. Total received message complexity is O(E), since each node receives message fron its neighbors, the total of received message is 2*E. The parameter update complexity is also O(E), since for each received message, we update the power cost to sink. Total sent message complexity is O(V), since each node sends periodic HELLO messages. The space complexity of the algorithm is O(E), since each node maintains a neighbor table which includes some data for each neighbor.

If we are on node A, we simply compute D_A(sink) = min{cost(A,X) + D_X(sink)} for each neighbor X. "D" stands for the distance of a node. The neighbor X which corresponds to minimum D_A(sink) is selected as the parent.

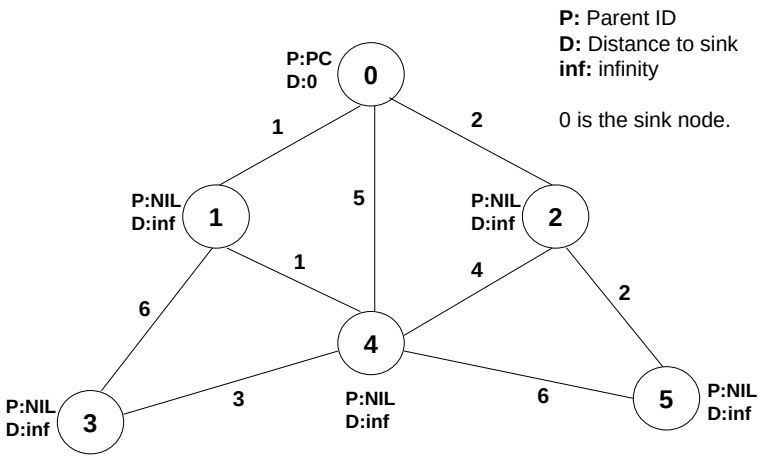


Figure 5.3: Initial configuration of the example graph. The edge costs are formed using the power estimation derived from RSSI values (by computing 1/RSSI).

Note that if a node failure occurs, neighbors detect it and remove the corresponding node from their neighbor tables. The node failure is confirmed if a HELLO message from the neighbor cannot arrive for three times in a row. Doing this prevents a node from sending its data to a non-existent location, which would lead to a disconnection of the network.

Look at the examples in Figures 5.3 and 5.4. In our algorithm, the edge costs are formed using the power estimation derived from RSSI values (by computing 1/RSSI). For example, the edge cost between node 0 and node 1 is formed after node 0 broacasts HELLO message and node 1 computes the link cost from the RSSI value extracted from that HELLO message. In Figure 5.3, the initial configuration of nodes along with their link costs are given. After parameters span the whole network, each node knows the minimum distance and parent to reach the sink node. The final configuration is in Figure 5.4.

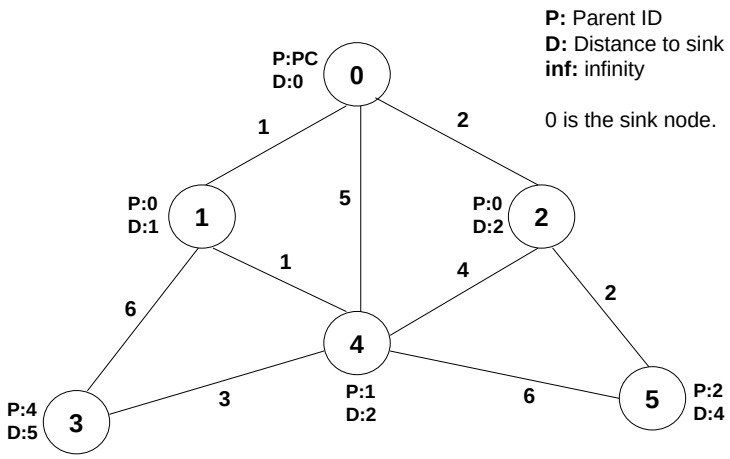


Figure 5.4: Final configuration of example graph after applying our algorithm.

Here is the code snippet written in nesC for Algorithm 1:

```
minimumDistance = FLT_MAX; // Maximum value of a float variable
parentNodeID = 255\; // NIL value for the parentNode

// Iterate through all neighbors
for (i = 0, j = 0;i < ROUTE_TABLE_SIZE;i++)
{
    pNbr = &NeighborTbl[i];

    if (!(pNbr->flags & NBRFLAG_VALID)) continue;
    if (pNbr->parent == TOS_LOCAL_ADDRESS) continue;
    if (pNbr->parent == ROUTE_INVALID) continue;

    if(pNbr->powerCostToSink != FLT_MAX
    && minimumDistance < ( pNbr->powerCostToSink + (1.0f / pNbr->avgReceiveRssi) ))
    {
        minimumDistance = pNbr->powerCostToSink + (1.0f / pNbr->avgReceiveRssi);
        parentNodeID = pNbr->id;
    }

    j++;
```

```
}

return parentNodeID;
```

5.2.2 Transmit Power Adjustment

According to RSSI values obtained from data packets coming into selected parent, the transmit power to the selected parent is adjusted around some threshold value. This threshold value was determined by us while doing the RSSI-distance measurement experiment (see Section 6.2.1.1). The value will converge to a value after some broadcasts such that when the RSSI value decreases some more from the threshold, the receiving node becomes unable to receive any more packets.

The adjustment can only be done if the same parent is selected both in the current execution and in the previous execution (60 seconds before).

Look at **Algorithm 2**

Algorithm 2: Transmit power adjustment using data RSSI values.

Input: Data RSSI value and the power level of the node. We are on node *A*.

Output: Power level is decreased or increased. We are on node *A*.

```
/* We check that whether the parent has changed, if it
   changed, then we reset the power level to the default.  The
   sink node always sends and forwards the data to the PC, so
   it does not require transmit power adjustment            */
```

if *A!= SINK-NODE and oldParent == newParent* **then**

```
    /* dataRssi value is updated from the HELLO message
       received from the parent.  If the selected parent node
       remains the same after some time, the power level will
       converge to the desired value.                        */
```

if *(dataRssi value is updated) and dataRssi* $\geq$ *DATA-RSSI-THRESHOLD* **then**

if *powerLevel != MIN-LEVEL* **then**

powerLevel -= 1;

end

end

else

if *powerLevel != MAX-LEVEL* **then**

powerLevel += 1;

end

end

end

else

powerLevel = DEFAULT-POWER-LEVEL;

end

For the time and space complexities of the transmit power adjustment algorithm, we analyze the local cost or the total network cost separately. The local cost for both time and space complexities are O(1). Because the algorithm does

not have any iteration (the time complexity), and the allocated space is constant (the space complexity). For the total network cost, since these computations are done in each node, the time and space complexities are O(V) where V stands for the number of nodes (vertices) in the network.

Here is the code snippet written in nesC for Algorithm 2:

```
if (pNewParent)
{
     if(pOldParent)
     {
         // If we are not on sink node and if the previously and currently selected parents are the same
         // Then we can adjust the transmission power
         if(TOS_LOCAL_ADDRESS != 0 && pNewParent == oldParent)
         {
             if(oldParent->avgDataSendRssi > RSSI_THRESHOLD)
             {
                 if(currentPowerLevel != MIN_LEVEL)
                 {
                     currentPowerLevel--;
                 }
             }
             else
             {
                 if(currentPowerLevel != MAX_LEVEL)
                 {
                     currentPowerLevel++;
                 }
             }
         }
         else
         {
             currentPowerLevel = DEFAULT_POWER_LEVEL;
         }
     }

     else
     {
         currentPowerLevel = DEFAULT_POWER_LEVEL;
     }

     // atomic is a keyword used in nesC. Atomic blocks are used to prevent race conditions.
     atomic
     {
          // Assign the new parent to the global variable we used to store the parent
```

```
            gpCurrentParent = pNewParent;
        }
    }
```

5.3 Discussion

There is an enhancement for our algorithm, which may make the network last longer.

Distributing the total energy-spent evenly over the sensor nodes makes the overall network last longer, since we extend the life-time of the first node to die.

This feature can be appended to our algorithm easily. We can store a list of neighbors along with their power costs (*ID*, *cost*), and sort this list with respect to the power costs. We should limit the size of the list, so that neighbors with high costs are unlikely to be selected. After that, we can iterate through the sorted list by enforcing a condition such that if the number of children of a node exceeds some threshold value, it will not be selected as a parent. In this case, we advance to the next candidate in the sorted list until we find a candidate satisfying the children count or we can select the one with the minimum children count to simplify the process.

It is not certain that whether this addition to the algorithm increases energy efficiency or not, so we did not append it to the current configuration.

This chapter covered our proposed multihop algorithm and its implementation details. The algorithm was divided into pieces where each piece was responsible for completing a task. Together, these tasks serve the goal of our algorithm which is to conserve energy by choosing an energy-efficient path and adjusting transmit powers of sensor nodes.

Chapter 6

Deployment and Experiments

In this chapter we first illustrate how the deployment of a sensor network has been done. Then we report the results of conducted experiments related to the deployment and the sensor characteristics which have some significance in the multihop algorithm. All the experiments are done with MICA2 sensor motes.

6.1 Deployment

Deployment of the sensors are done in the half of the 4th floor of the Engineering Building of Bilkent University. The MICA2 sensor plugged with the sensors boards are scattered across the area so that they form a wireless sensor network in which each node can reach the sink-node via one or more hops.

The Figure 6.1 shows the deployed sensor network with the location of each individual sensor. The ID of the sink node is 0. The deployed environment reflects the indoor behavior. In the network, there are obstacles between the sensor motes which tend to increase the packet loss count because of the multipath effects. The routes to the base station is established using our proposed multihop routing algorithm described in Chapter 5.

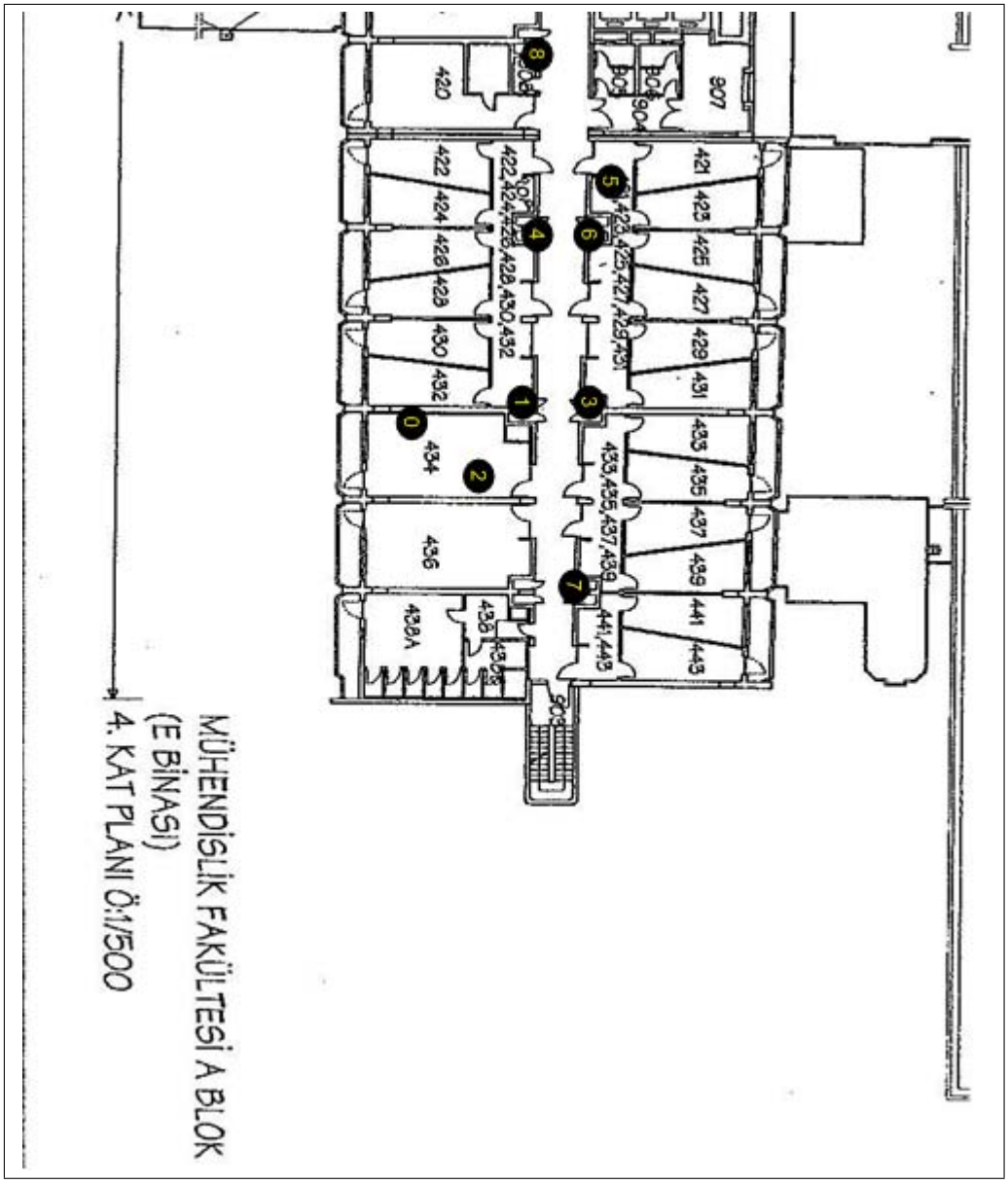
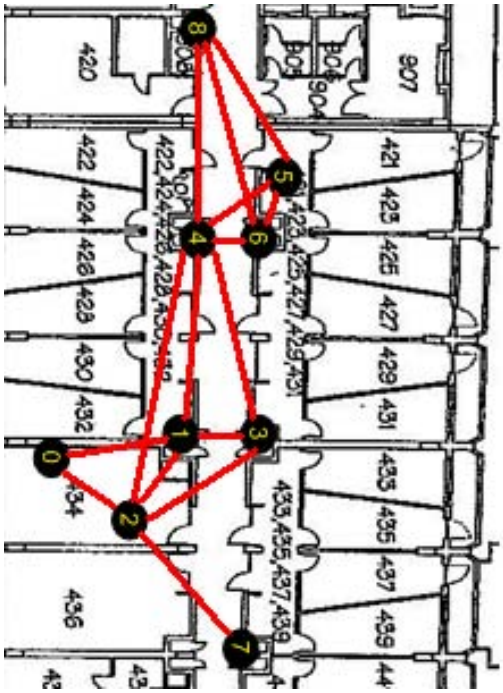


Figure 6.1: Part of 4th floor plan of Engineering Building, Bilkent University.

The topology changes dynamically due to the changes in the quality of wireless links. Routing paths change frequently as well. Example topologies of the deployment are presented in Figures 6.2, 6.3, and 6.4.



NodeID	Location
0	BerkBilgisayar
1	OfisDuvarArkasi
2	BurcuMasa
3	OfisKarsisi
4	YanginSondurucuYani
5	SwitchDolabiUstu
6	YanginSondurucuKarsisi
7	3DTVOdasiYani
8	DekanliginYani

Figure 6.2: On the left: example topology 1 is shown in the floor plan; On the right: the id→location correspondence.

6.2 Experiments

The experiments are classified into two categories. These are:

- **Deployment-Independent Experiments:** These experiments have been done in small-scale, which focus on characteristics of sensor motes, and the RSSI improvement applied in the proposed multihop routing algorithm.
- **Deployment-Based Experiments:** The results of these experiments include the statistical measurements of temperature, light, noise and the battery power at various locations. They also include the reply times to issued

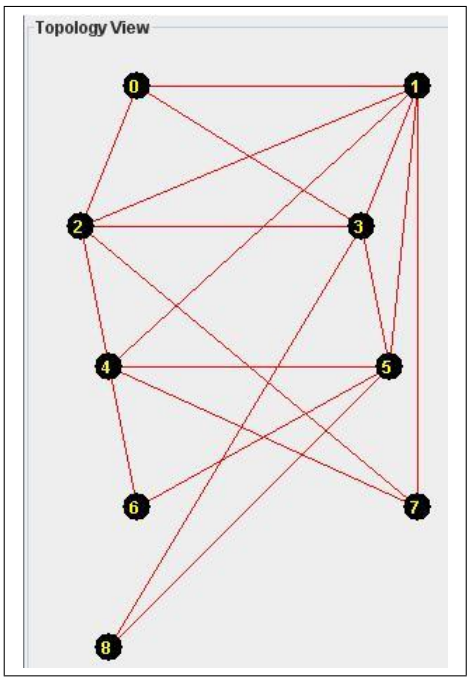


Figure 6.3: Example Topology 2.

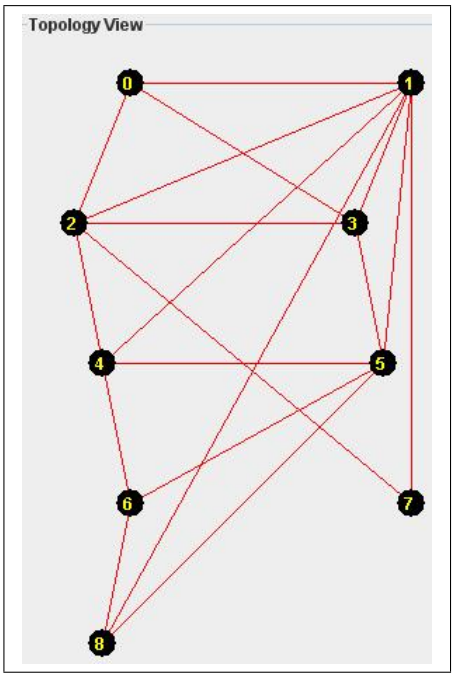


Figure 6.4: Example Topology 3.

queries, and the packet loss percentages for hard-to-reach sensors.

6.2.1 Deployment-Independent Experiments

6.2.1.1 RSSI Related Experiments

RSSI values have a crucial role in constructing the routes in our multihop algorithm, since each node uses them for determining the parent node. As mentioned in Chapter 5, the RSSI values are scaled between 0 and 255 for saving some space in the packet payload. The experiment results are also based on these scaled values.

In the first experiment, the change of RSSI values is observed with respect to the distance which is illustrated in Figure 6.5. The experiment was done in the office which is an indoor environment. Multipath effects in the office caused the signal to fluctuate. Fluctuations in the signal can cause varying RSSI values for the same distance. Distance estimation using this measurement is error-prone, because the RSSI value at 100 cm is lower than the one at 150 cm even though 100 cm is nearer than 150 cm.

In our proposed algorithm, taking the average of the last 10 measurements is suggested to overcome this problem. Figure 6.6 shows the outcome of applying taking-average method. At fixed distances, the amount of time between two RSSI measurements is 10 seconds. Distance estimation becomes more realistic, because RSSI value becomes inversely proportional with the distance.

Another experiment was done to show that the fluctuations in the received signal was minimized. Figure 2.7 in Chapter 2 proves the unreliability of RSSI values, since they vary significantly even the distance does not change. We managed to overcome that issue by taking the average method.

The nodes were placed into the lab office. Figure 6.7 shows the configuration for this experiment. Node 0 is the sink-node and connected to the PC, hence is mains powered, whereas node 1 and node 2 are battery-powered. There were

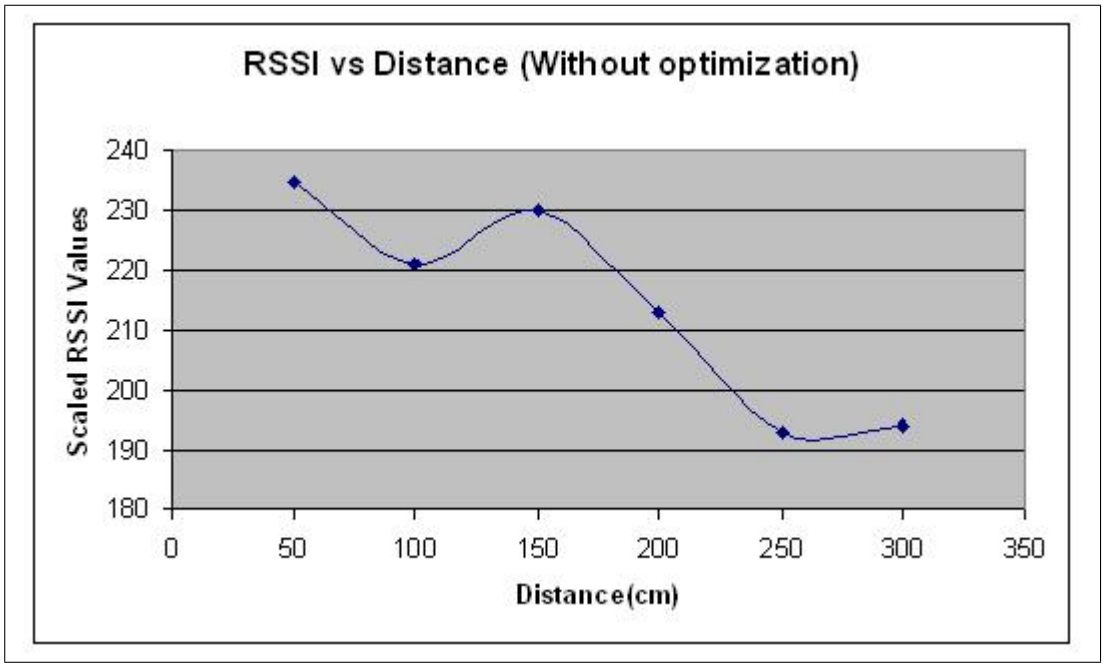


Figure 6.5: RSSI vs distance without any optimization.

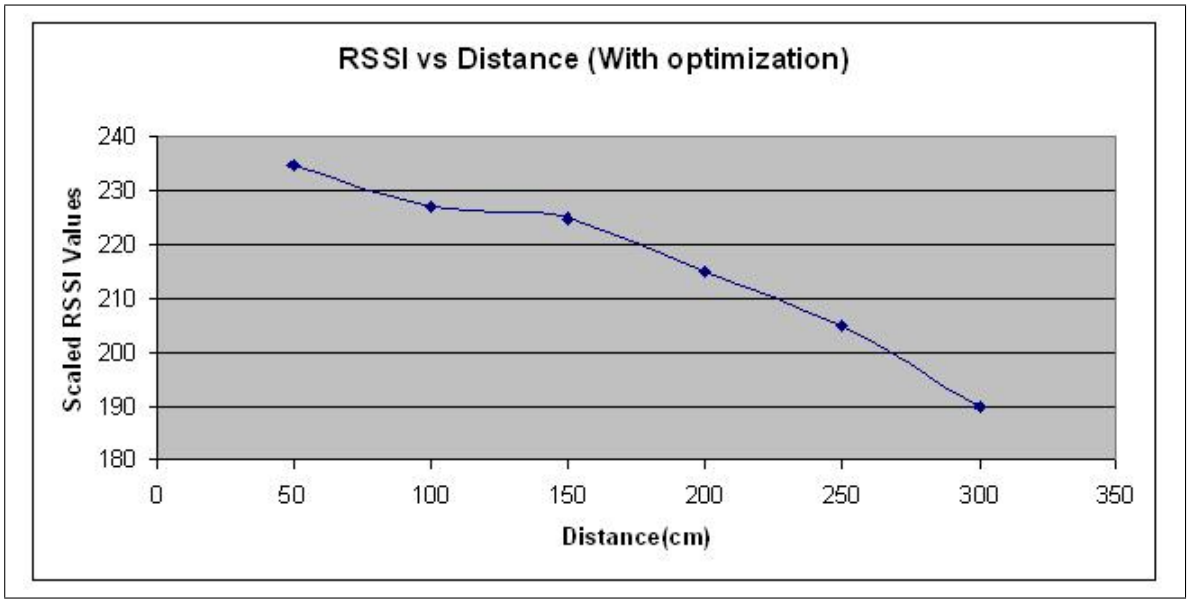


Figure 6.6: RSSI vs distance with the optimization.

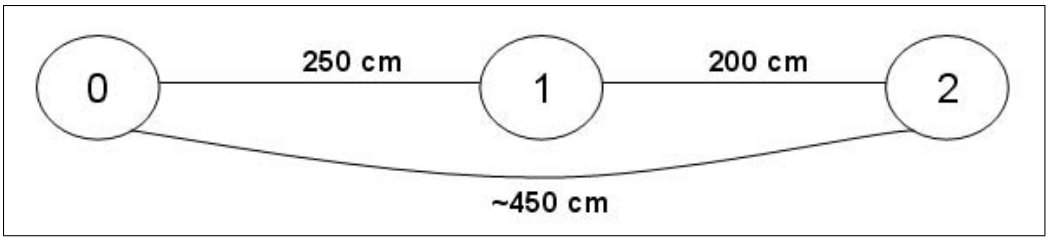


Figure 6.7: Experiment configuration for fluctuation minimization in RSSI.

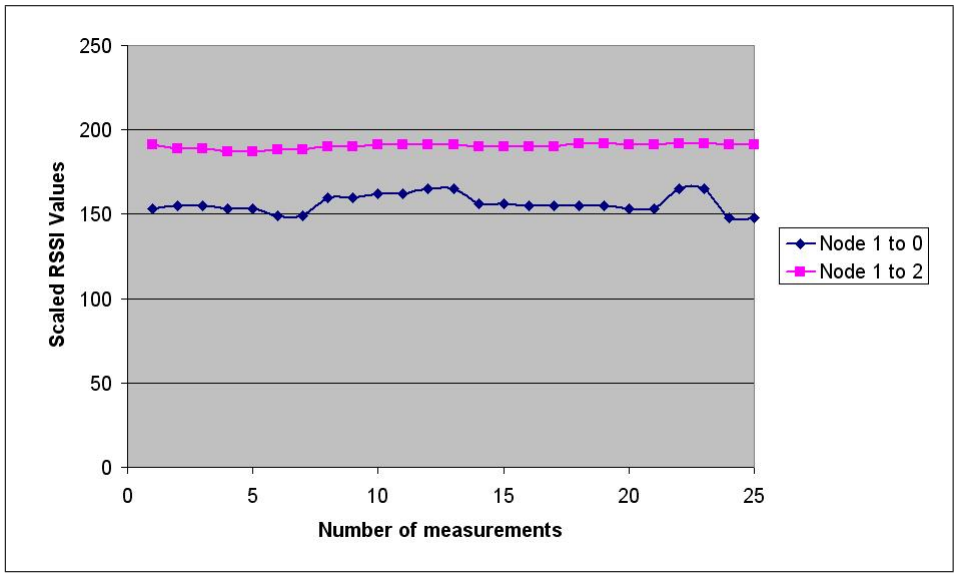


Figure 6.8: RSSI measurements measured on other nodes when node 1 sent packets. Each RSSI measurement is calculated by taking the average method.

obstacles between node 0 and node 2.

The established data path were:

- Node 1 sent its data to node 0.
- Node 2 sent its data first to node 1, then the data is forwarded to node 0.

The results are depicted in Figures 6.8 and 6.9. The results show that taking the average of the last 3 measurements minimized the fluctuation of RSSI values. There are total of 4 curves plotted in 2 figures:

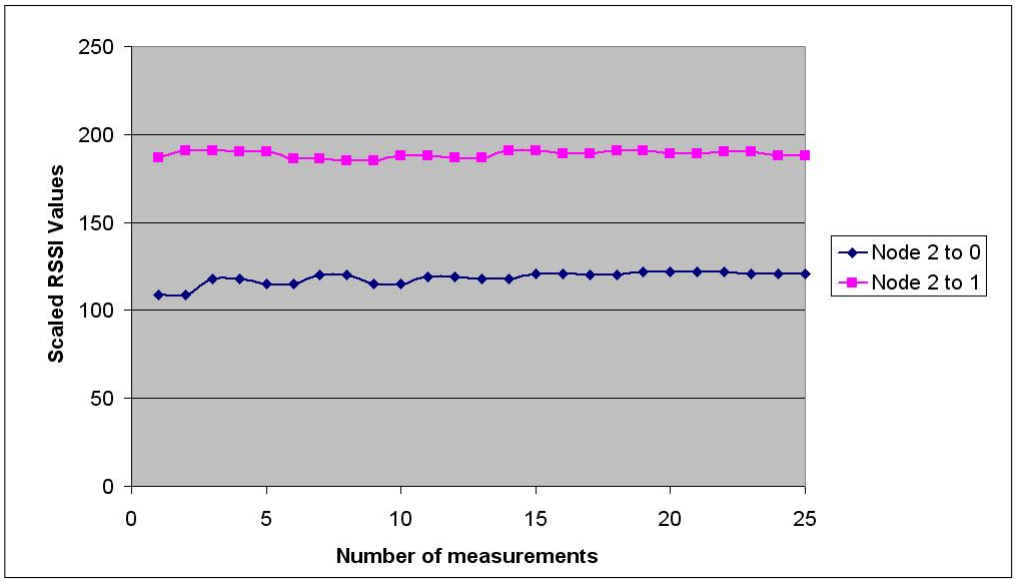


Figure 6.9: RSSI measurements measured on other nodes when node 2 sent packets. Each RSSI measurement is calculated by taking the average method.

- **Node 1 to 0:** RSSI value measured on node 0, while node 1 was sending control messages to node 0.
- **Node 1 to 2**: RSSI value measured on node 2, while Node 1 was sending control messages to node 2.
- **Node 2 to 0:** RSSI value measured on node 0, while node 2 was sending control messages to node 0
- **Node 2 to 1:** RSSI value measured on node 1, while node 2 was sending control messages to node 1

The main point in the results is the minimization of variance, which is preferrable for consistent estimation of distance, and also helps the estimation of the power consumption of the link.

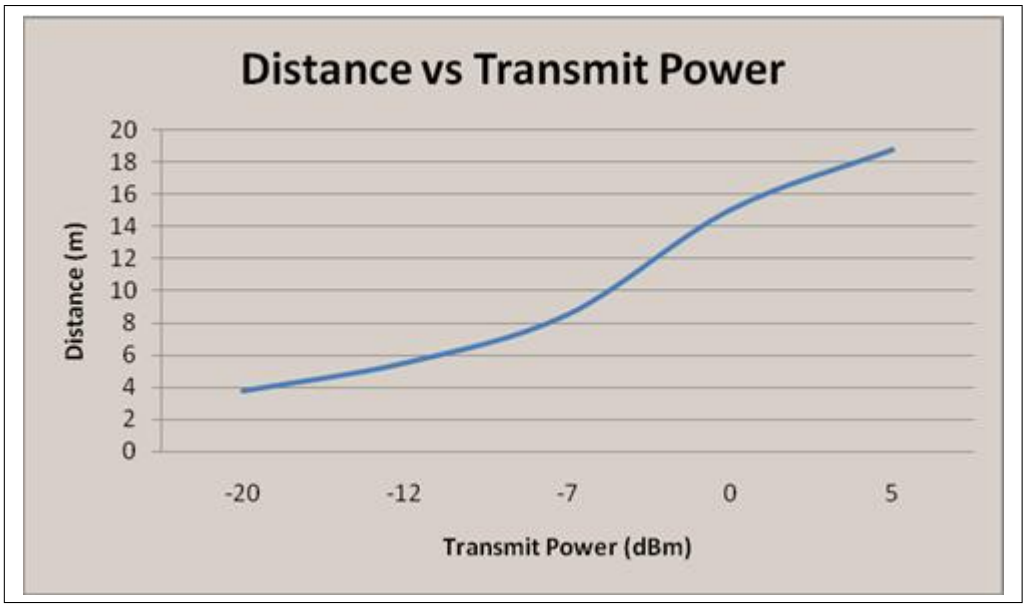


Figure 6.10: Distance vs Transmit-Power.

6.2.1.2 Transmit-Power Related Experiment

Transmit power is critical in our system. Each node adjusts its transmit power in such a way that the node's coverage range becomes just enough to contain the parent node. Our aim in this experiment was to observe how the transmission range of the radio changes with the transmission power.

The experiment was done with two sensor motes. One of them was connected to the PC (sink node) and the other was mobile. The mobile node was constantly transmitting bulk data to the sink node in a direct line of sight manner. The led lights of the sink node were toggling as long as it was in the mobile node's transmission range. We measured the distance at the point where the toggling of the leds stopped. We did this measurement for each transmission power ranging from -20 dBm to 5 dBm. The result is depicted in Figure 6.10. Consequently, the transmission range increases almost linearly with the transmission power (in dBm).

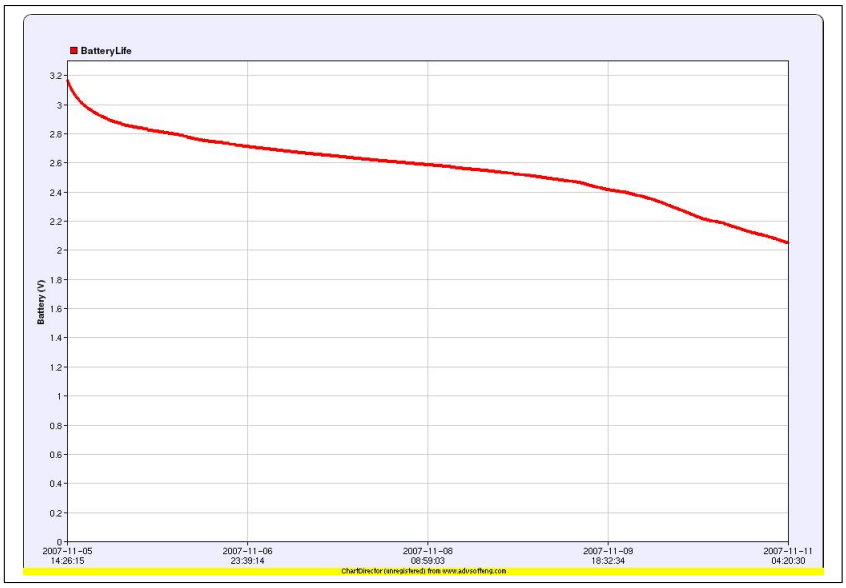


Figure 6.11: Battery life of a MICA2 node with the radio always active and minimum processing power consumed.

6.2.1.3 Battery Life Experiment

The MICA2 sensor motes work with 2xAA batteries. The life-time of these batteries is observed whether the sensor motes are capable of lasting in a long-term deployment. During this experiment, a sensor mote is configured to consume minimum processing power. However, the sensor mote constantly transmitted a dummy packet for spending its energy, because the main source of the power consumption in wireless sensor networks is the radio transceiver. In the experiment, the number of packets transmitted was approximately 16078800 and the size of the each packet was 29 bytes, so the total size of the transmitted data was approximately 444.684 MB (until battery dies) which is quite large for a tiny and battery powered sensor mote. For example, if a node sends one packet per minute which is 29 bytes, but does nothing else to spend its energy, then the node will last approximately about 117 days.

The experiment was done with two sensor motes, which is similar with the transmission-range experiment. The mobile node sent 1 in every 100 packets to

the sink node, while the other 99 packets were sent to a non-existent destination. Our PC stored these measurements in a mySQL database server and we plotted the stored values. The value includes the voltage measurement.

The battery voltage plot is shown in Figure 6.11. The battery became unfunctional after the volt value decreases to around 2 volts. The sensor mote was alive for almost six days. With the energy conversation policy of our multihop routing protocol, a wireless sensor network consisting of these motes can last for a long time.

6.2.2 Deployment-Related Experiments

6.2.2.1 Network Quality Experiments

There are 9 sensor motes deployed in the system. When a user wants to send a query, the queries disseminate from the sink node to the destined node which replies back the results using the multihop routing algorithm. The time and the frequency of the replies are submitted by the user. We conducted two experiments. Each of these experiments were done with 2 different duty cycle modes which are duty cycle mode 4 (%5.61 idle listening ratio) and mode 0 (%100 idle listening ratio).

In the first experiment, we queried the sensors so that there would be 1 reply after 10 seconds. It is done by assigning the duration value, and sampling rate value to 10 in the "Sampling Query" page. We queried the 8 different sensor motes, 10 attempts per each. We did not query the sink node, because its delivery is guaranteed since it does not require wireless communication. The round-trip delay, the time between the query is issued and the query is replied back, is calculated for each of the attempts. For each of these attempts, we also recorded the routing paths that the packets followed in order to reach the base station. Table 6.1 and 6.2 show the round-trip delays and the routing paths respectively for duty cycle mode 4. The topology of the network using Table 6.2 is depicted in Figure 6.12. We do not include these tables for duty cycle mode 0, since

Attempt	Node ID							
	1	2	3	4	5	6	7	8
1	10.829	10.607	11.086	10.837	11.781	11.069	10.844	12.06
2	10.846	10.619	11.068	11.091	12.104	no	11.137	12.643
3	10.88	10.654	11.069	no	no	11.098	10.843	no
4	11.228	10.609	11.059	10.841	11.313	11.317	10.845	12.056
5	10.843	10.614	no	10.834	11.788	no	10.85	12.515
6	11.088	10.622	11.074	no	no	11.077	11.139	no
7	10.815	10.653	11.661	11.102	12.362	11.732	11.098	12.512
8	10.842	10.608	11.107	11.124	11.347	11.069	10.838	no
9	10.839	10.648	11.078	no	no	11.547	10.845	12.488
10	11.13	10.685	11.351	11.135	11.551	11.084	11.096	no
Average	10.834	10.632	11.173	10.995	11.749	11.249	10.954	12.379

Table 6.1: Query round-trip delays (in ms) for nodes for the first experiment which was done in duty cycle mode 4 (with 5.61% idle listening).

the routing paths in both modes are same, and our goal is to show different outcomes between these two modes. For the sake of comparison, we present average response times and packet delivery ratios measured on both duty cycle modes.

In Table 6.2, we observe that the paths are consistent and does not change. This means that we managed to overcome the disadvantages of indoor environments like the fluctuations of signals, since these paths are determined with link costs derived from RSSI measurements.

Average response times for duty cycle mode 4 are plotted in Figure 6.13. According to that figure, the response of node 8 arrives latest. The reasons is that the increase in the number of hops postpones the delivery. Average response times for duty cycle mode 0 are plotted in Figure 6.14. As we increase the idle listening ratio, the packets arrives faster, but each node spends more energy, so there is a trade off between the energy consumption and the delay.

Attempt	Node ID							
	1	2	3	4	5	6	7	8
1	1→2→0	2→0	3→1→2→0	4→2→0	5→4→2→0	6→4→2→0	7→2→0	8→5→4→2→0
2	1→2→0	2→0	3→1→2→0	4→2→0	5→4→2→0	no	7→2→0	8→5→4→2→0
3	1→2→0	2→0	3→1→2→0	no	no	6→4→2→0	7→2→0	no
4	1→2→0	2→0	3→1→2→0	4→2→0	5→4→2→0	6→4→2→0	7→2→0	8→5→4→2→0
5	1→2→0	2→0	no	4→2→0	5→4→2→0	no	7→2→0	8→5→4→2→0
6	1→2→0	2→0	3→1→2→0	no	no	6→4→2→0	7→2→0	no
7	1→2→0	2→0	3→1→2→0	4→2→0	5→4→2→0	6→4→2→0	7→2→0	8→5→4→2→0
8	1→2→0	2→0	3→1→2→0	4→2→0	5→4→2→0	6→4→2→0	7→2→0	no
9	1→2→0	2→0	3→1→2→0	no	no	6→4→2→0	7→2→0	8→5→4→2→0
10	1→2→0	2→0	3→1→2→0	4→2→0	5→4→2→0	6→4→2→0	7→2→0	no

Table 6.2: Query reply paths that the packets followed for the first experiment which was done in duty cycle mode 4 (with 5.61% idle listening).

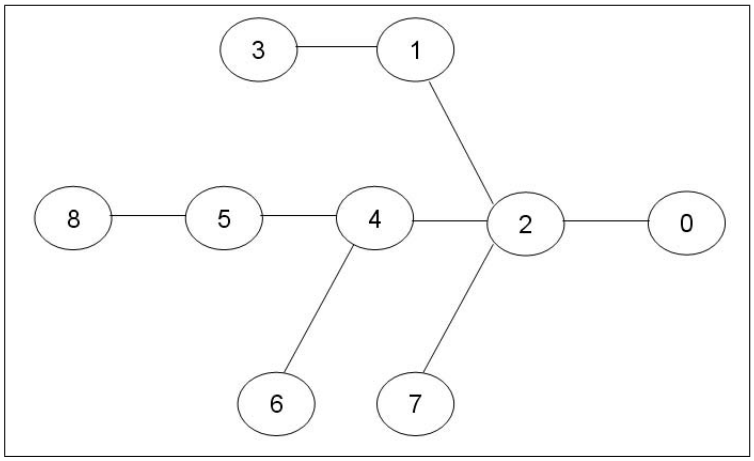


Figure 6.12: Topology of the network according to the routing paths in Table 6.2.

The location of the sensor nodes affect the delivery rate of packets. The originator node may be far from the its neighbor making the wireless link unreliable. Obstacles and interference of other wireless devices can lower the rate of delivery as well. Figure 6.15 shows the packet delivery ratio for duty cycle mode 4. If we increase the idle listening ratio, packets are less likely to be lost as well. Figure 6.16 shows the packet delivery ratio for duty cycle mode 0. Since we managed to preserve approximately 20 times more energy by decreasing the idle listening ratio to %5.61, it is worth losing some packets.

In the second experiment, our goal was to see the packet delivery ratio under heavy traffic. Sensor nodes were sampled in a way that they had to reply once in every 5 seconds for duration of 500 seconds. This means that they are responsible for sending 100 replies. Sensor nodes were not sampled at the same time, we waited around 500 seconds for each node to finish their task. The packet delivery ratios for duty cycle modes 4 and 0 are given in Figure 6.17 and 6.18 respectively. The results resemble the ones in Figure 6.15 and 6.16, proving the trade off between the energy consumption and the packet delivery.

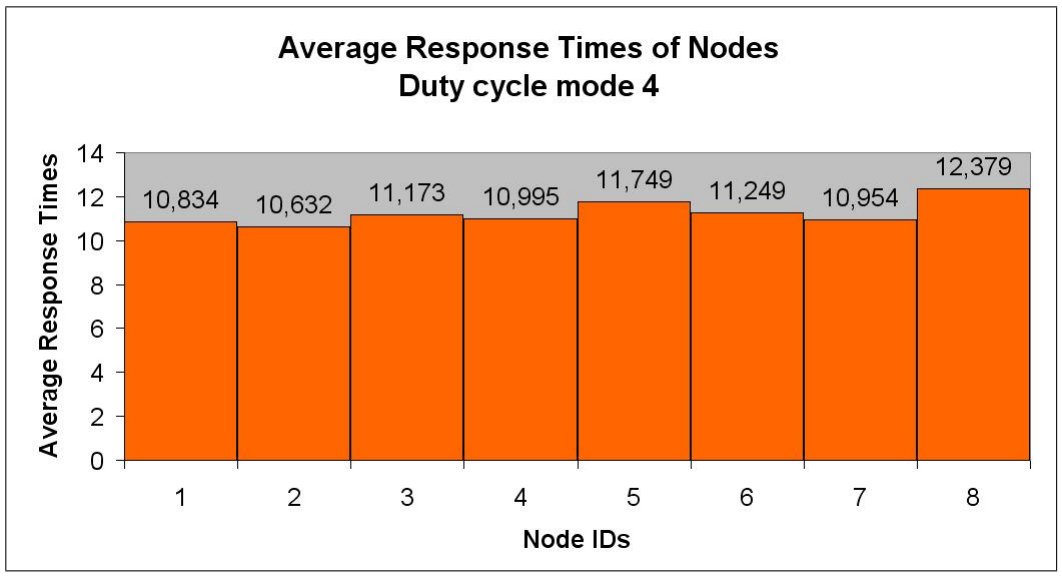


Figure 6.13: Average response times for the deployed nodes for the first experiment which was done in duty cycle mode 4 (with 5.61% idle listening).

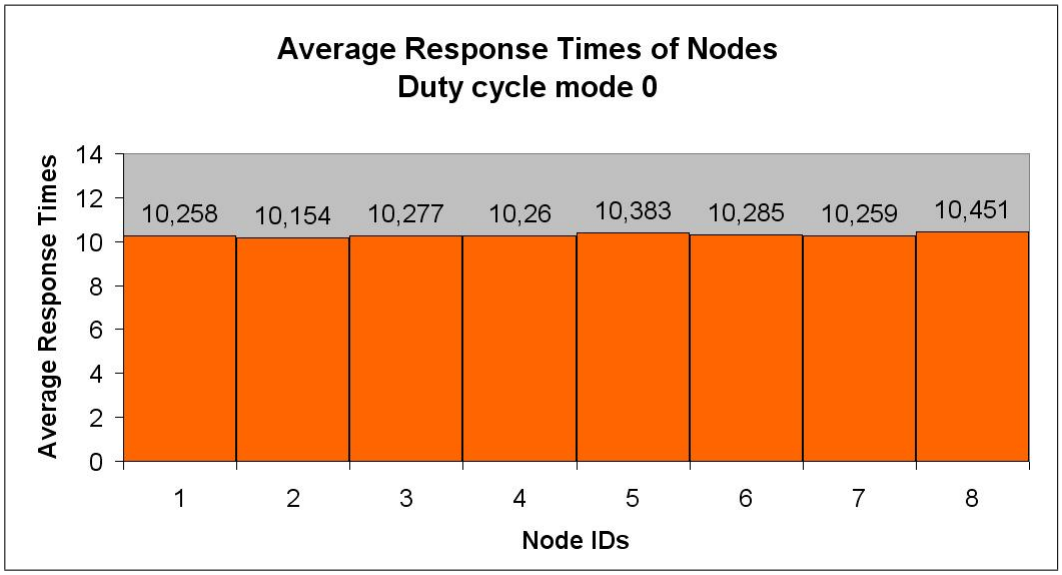


Figure 6.14: Average response times for the deployed nodes for the first experiment which was done in duty cycle mode 0 (with 100% idle listening).

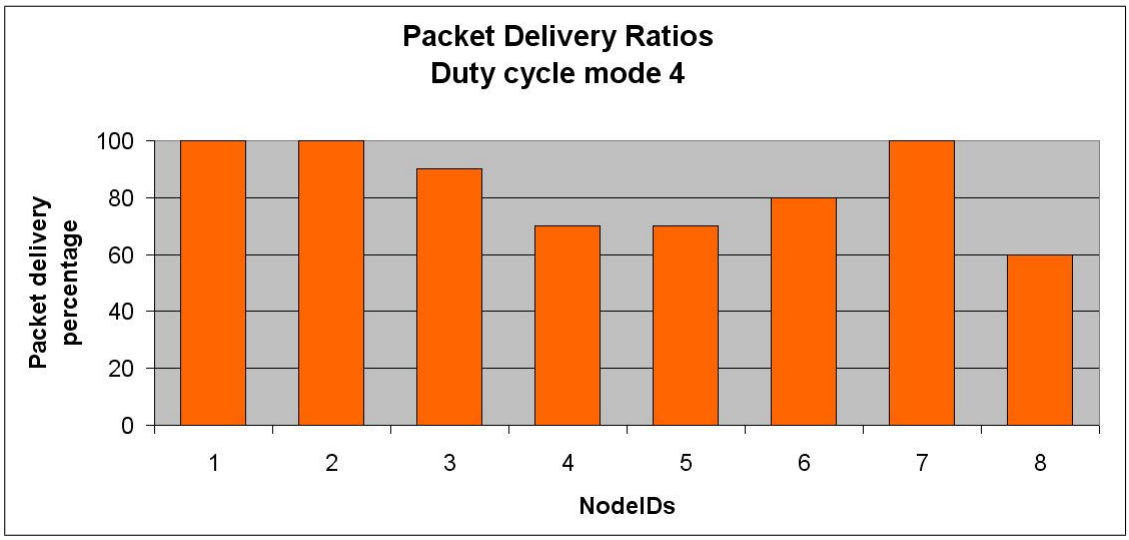


Figure 6.15: Packet delivery ratio for the deployed nodes for the first experiment which was done in duty cycle mode 4 (with 5.61% idle listening).

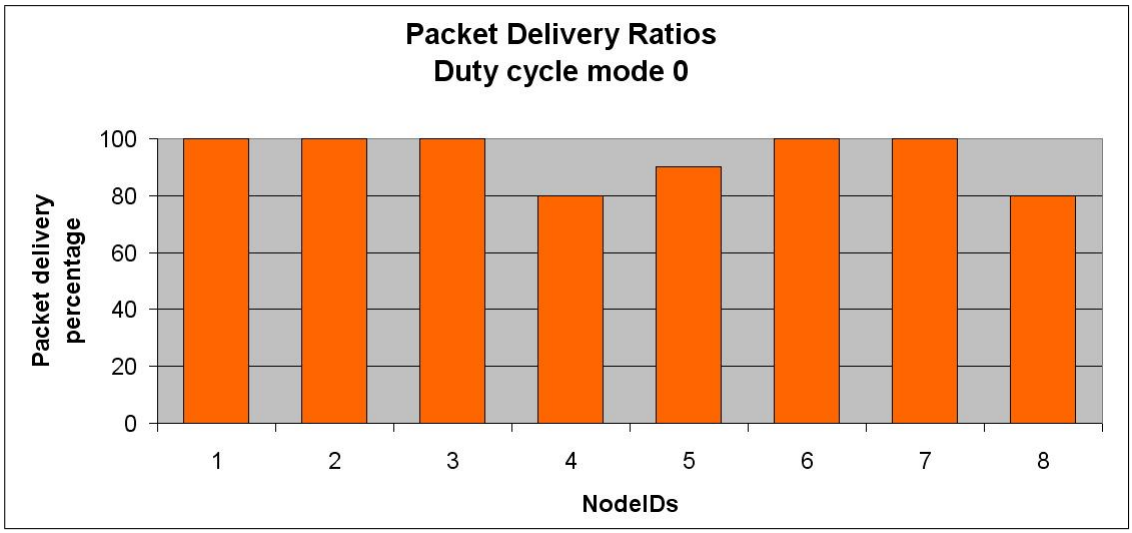


Figure 6.16: Packet delivery ratio for the deployed nodes for the first experiment which was done in duty cycle mode 0 (with 100% idle listening).

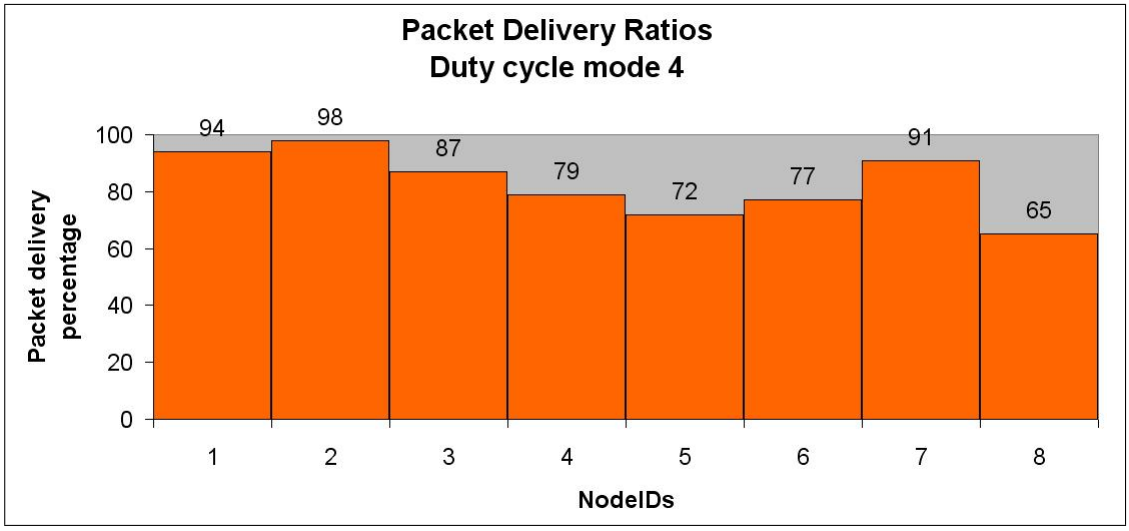


Figure 6.17: Packet delivery ratio for the deployed nodes for the second experiment which was done in duty cycle mode 4 (with 5.61% idle listening).

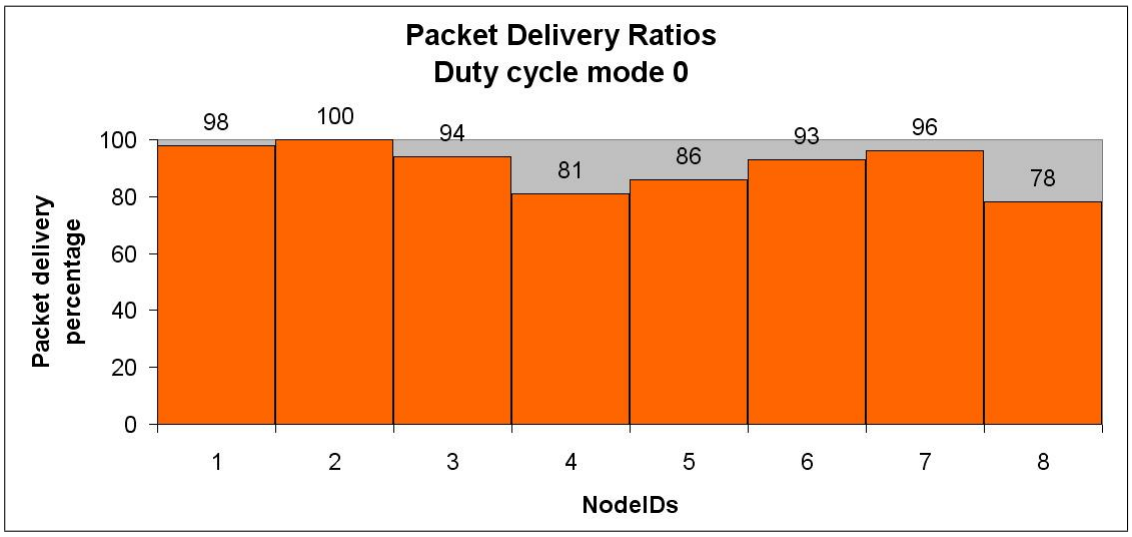


Figure 6.18: Packet delivery ratio for the deployed nodes for the second experiment which was done in duty cycle mode 0 (with 100% idle listening).

6.2.2.2 Dynamic Topology Scenario

A wireless sensor network has to be adaptive to the changes in the topology in case a node is displaced, meaning that it moved from a location to another location. It is important that the packets continue to arrive at the sink node after the topology changes.

We conducted an experiment to see how the system reacted to the changes in the topology. We changed the location of *node 4*. Changes in the routing paths are illustrated in Figure 6.19. The children of *node 4* in the original topology had to change their parents to adapt the new topology. Consequently, *node 6* changed its parent to *node 1*, and *node 5* changed its parent to *node 6*. Also, *node 7* chose *node 4* as the parent node, because *node 7* discovered that *node 4* had the minimum total cost to the sink node.

The experiment was done in duty cycle mode 0. Since the duty cycle has a negligible effect on the timing of determining the new topology, the results of this experiment do not change for other duty cycles.

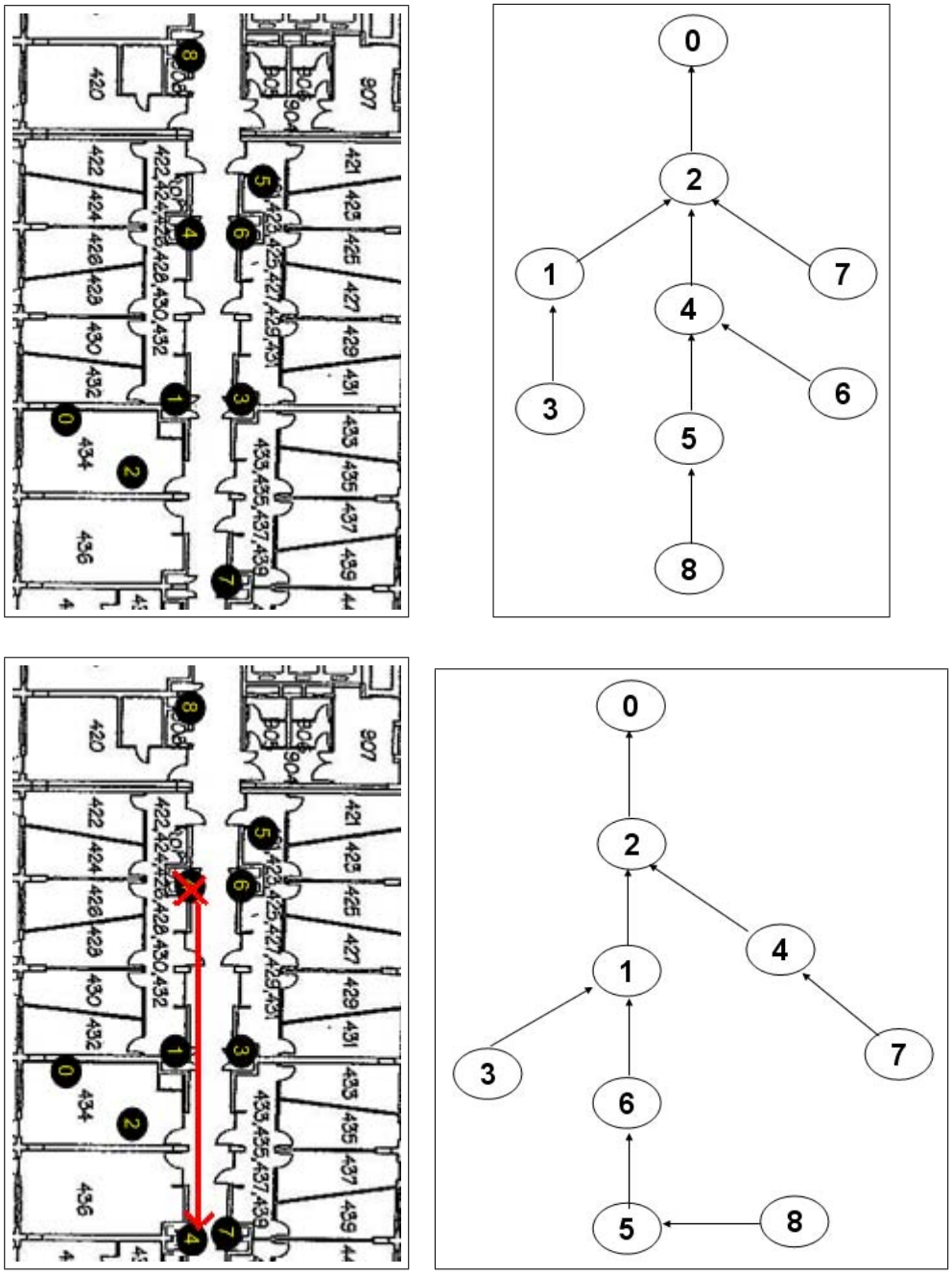


Figure 6.19: The upper part of the figure shows the original deployment (upper-left) and the routing paths of the original deployment (upper-right). The lower-left figure shows the modified topology, in which *node 4* is displaced to another location. The lower-right figure shows the routing paths of the modified deployment.

Attempt	Node ID	
	5	6
1	135	131
2	137	126
3	127	113
Average	133	123.3

Table 6.3: Adaptation times (in seconds) of *node 5* and *node 6* to the new topology.

The change in the topology primarily affects *node 5* and *node 6*, because in the original topology, they were the children of *node 4*. We had 3 runs for this experiment. Their adaptation times for each attempt along with their averages are given in Table 6.3. The adaptation times are reasonable for wireless sensor networks.

6.2.2.3 Statistical Experiments

The sensor nodes route the data to the sink node, and this data carries temperature, noise, light and battery energy measurements. These measurements are stored in a mySQL database for future observations. These stored measurements are displayed with graphical and textual representations. Daily, weekly, and average of these measurements are plotted for informing the user. This also makes the system easy-to-use for the administrator. For instance, the motes with weak batteries can be clearly detected, and these batteries are changed with the new ones to keep the system running properly.

Figures 6.20, 6.21, 6.22 and 6.23 show the daily temperature, noise, light and battery power patterns respectively for the date 8/7/2008 and sensor mote location "SwitchDolabiUstu" which has the node ID 5. The unit of the temperature is celsius and remaining battery energy is represented as volt values. These measurements may give us some idea what is going on in that environment. For example, we can guess how long the people were present in an office by looking at the light values, or at what times the air conditioner was working by looking

at the temperature values. The weekly distribution of the temperature and light values of the corresponding sensor mote location is illustrated in Figure 6.24 and Figure 6.25 respectively. Since the sensor motes cover half of the 4th floor, a more precise temperature value of that whole area can be obtained by taking the temperature mesaurements of all deployed motes. The bar chart in Figure 6.26 shows the average temperature per day of all nodes gathered through the week.

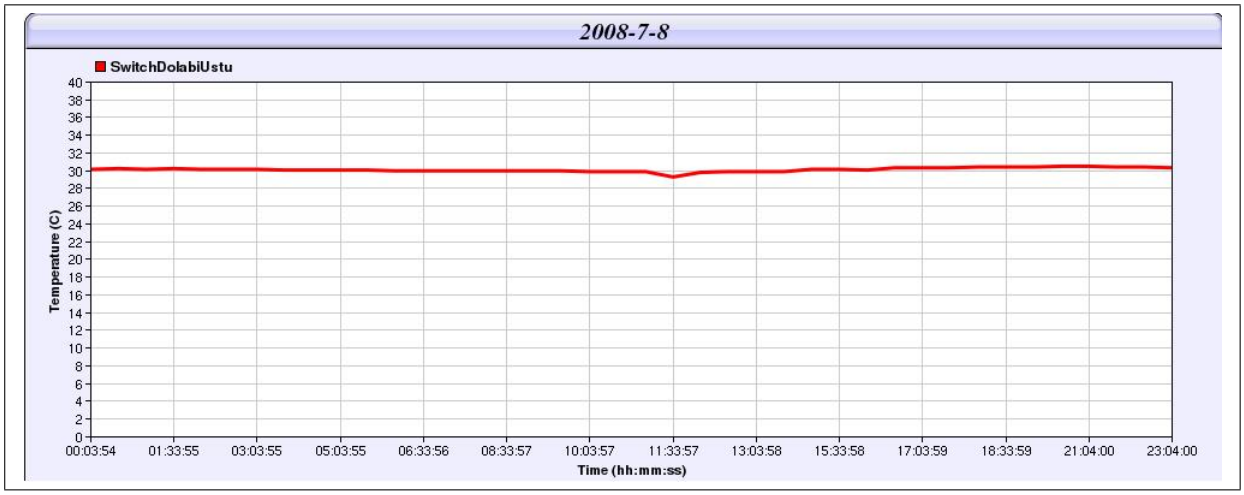


Figure 6.20: Daily temperature distribution for SwitchDolabiUstu.

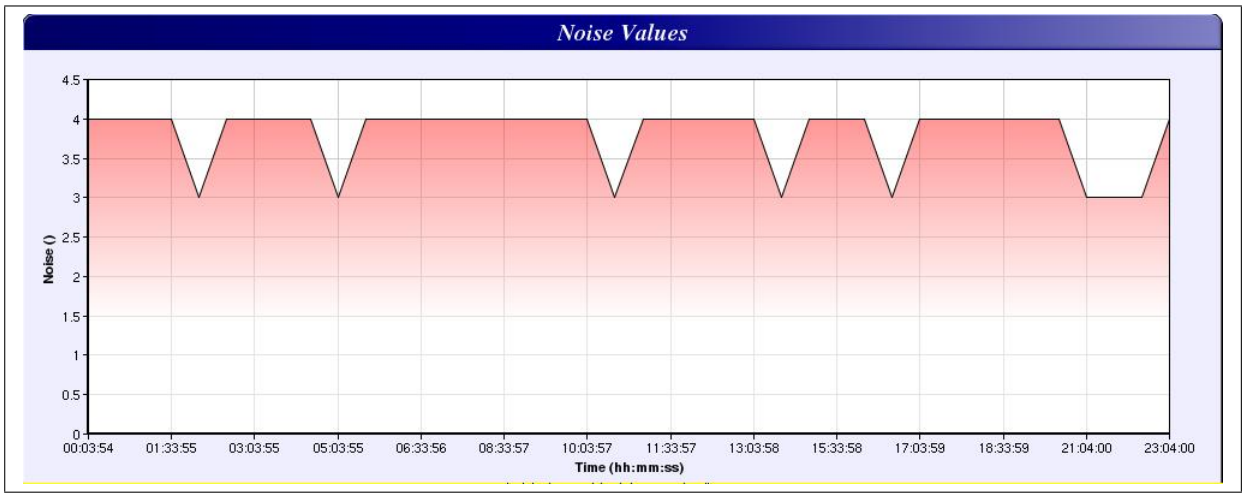


Figure 6.21: Daily noise distribution for SwitchDolabiUstu.

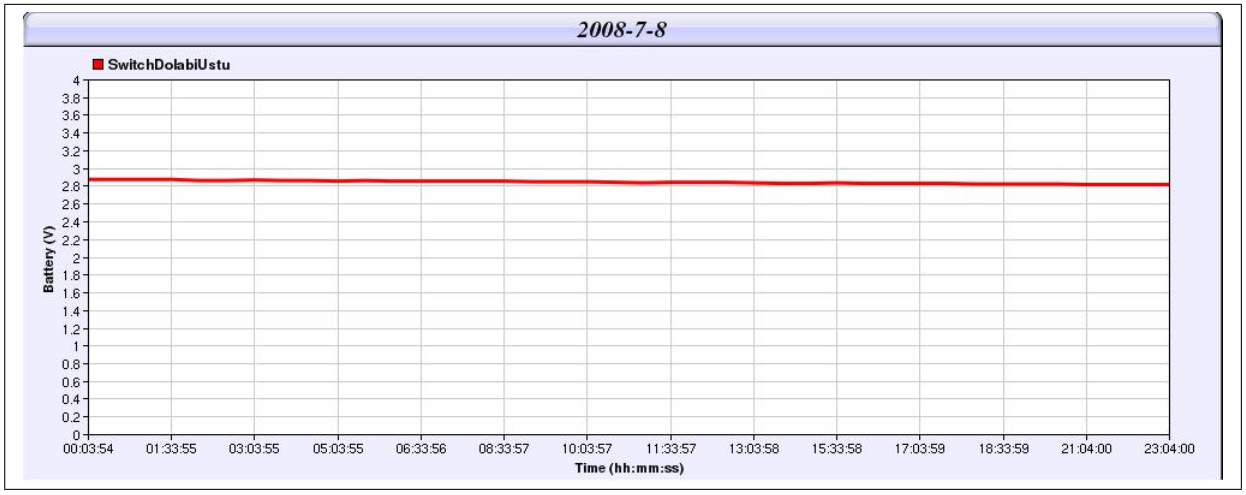


Figure 6.22: Daily battery energy distribution for SwitchDolabiUstu.

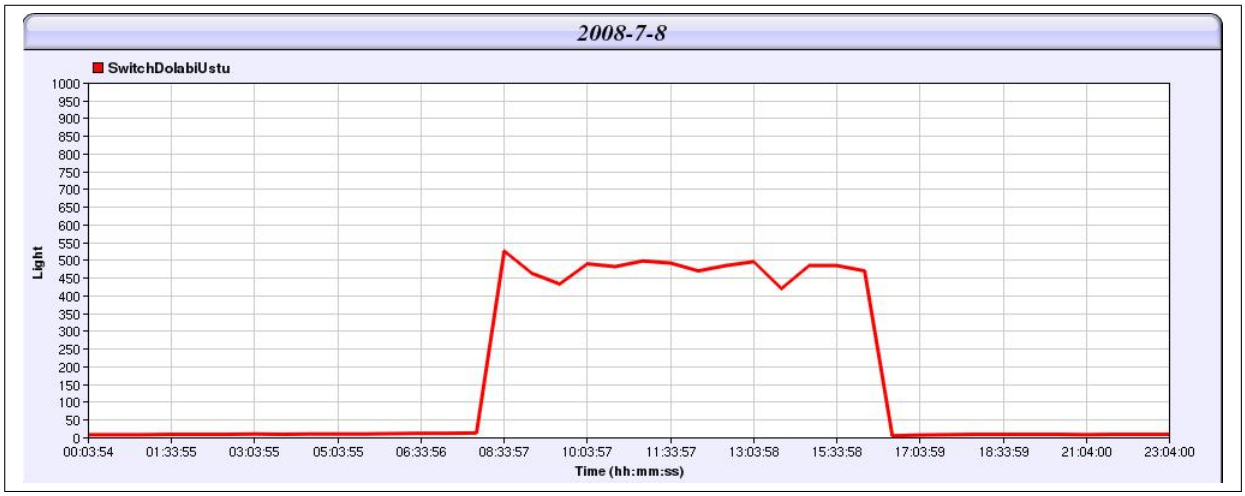


Figure 6.23: Daily light distribution for SwitchDolabiUstu.

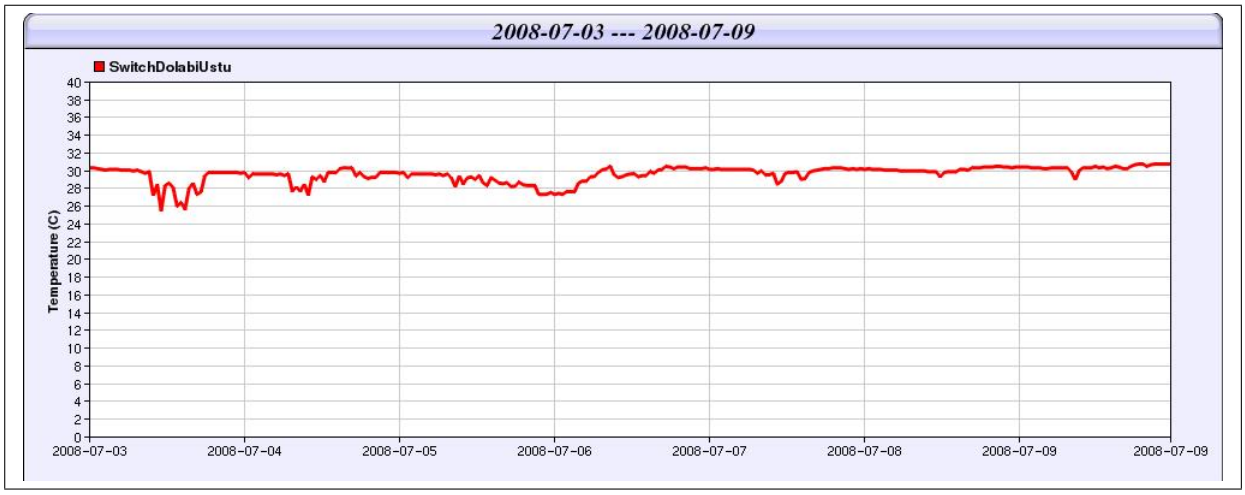


Figure 6.24: Weekly temperature distribution for SwitchDolabiUstu.

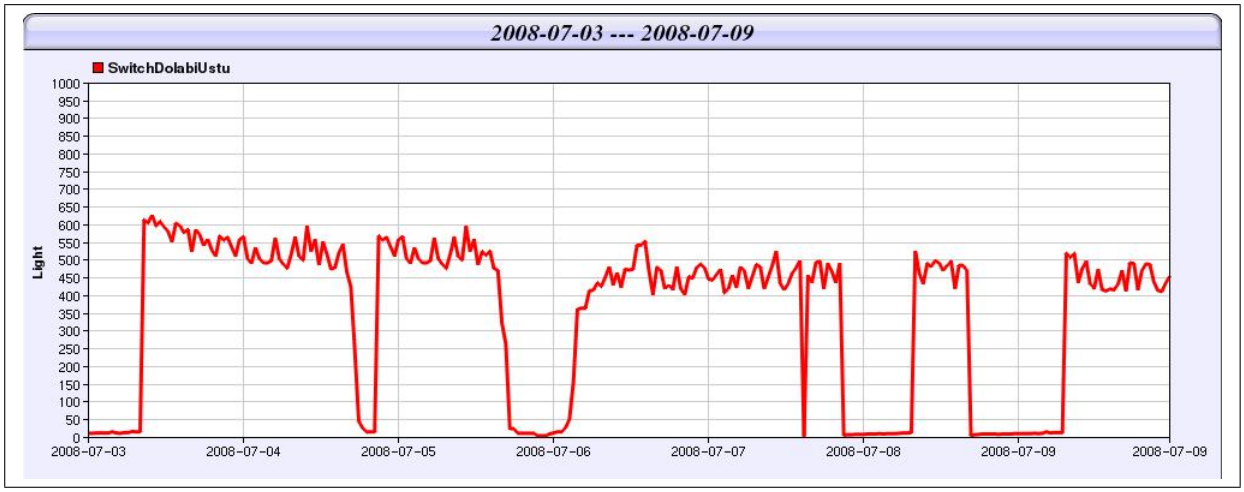


Figure 6.25: Weekly light values distribution for SwitchDolabiUstu.

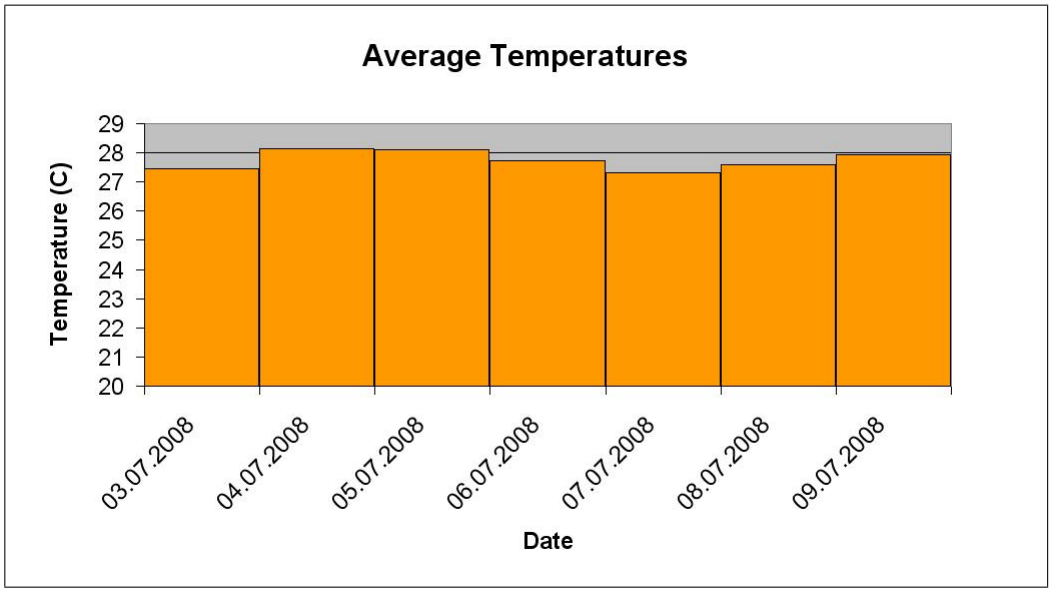


Figure 6.26: Average temperature values from all deployed nodes in a week per day.

In this chapter, we illustrated the deployment of the sensors. We conducted several experiments to show the characteristics of sensors and improvements of our multihop routing algorithm. The results show that our system works effectively and achieves a good performance on packet deliveries and response times. It is also proven to be useful for environment monitoring purposes.

Chapter 7

Conclusions and Future Work

Wireless sensor networks can be used for a variety of purposes. Monitoring the environment is one of them. Each sensor node is responsible for sensing and sending its monitored data to the base station via one or more hops. In this thesis, we presented an enviroment monitoring system along with the improvements for making the system long lived and increasing the system usability.

Durability of the system depends on the energy spent, so we tried to design a multihop routing algorithm such that each individual node spends less energy. We preferred the adjustment of transmit power, and the adjustment is done based on the link qualities and distances between the nodes. The link qualities are estimated using RSSI measurements. In the Chapter 2, we first investigated the previous work about proposed multihop algorithms, using RSSI as a distance estimation in indoors, and the transmit power concept. In the Chapter 5, we proposed a multihop routing algorithm using these concepts. We also decreased the percentage of idle listening of the radio to around 5% percent and this also contributes to energy conservation.

The main contributions of this thesis are: a new multihop routing algorithm is proposed and implemented for conserving the energy in wireless sensor networks, and a testbed was developed and installed and it was made functional in a building. Chapter 4 explains how this system is designed and implemented.

The system is made accessible through the Internet, offers the opportunity for the users to investigate it by submitting queries and seeing the cureent topology. The deployment of the sensor motes was done in the building. Furthermore, the measurements were taken to illustrate the effectiveness of the system. The reply times were fast even though the destination node was many hops away.

The system is designed in such a way that the only thing the administrator has to do is renew the weakened batteries in order to keep the system running. Adding new motes or removing motes in the system is easily done as well. The administrator page serves this purpose.

The system can be improved by migrating the code from TinyOS version 1.x to version 2.x in the future. This is because in the following years, TinyOS version 1.x may not be widely supported, and version 2.x will probably be extensively used for developing wireless sensor network applications.

We can also extend our multihop routing algorithm to be more energy-efficient. First of all, a sleep scheduling algorithm can be implemented, so that each sensor mote will go into the sleep mode when there is no task to do. This extension can improve the energy efficiency significantly. Moreover, a well-built data dissemination mechanism could help us to save more energy, since we need to disseminate the submitted queries throughout the wireless sensor network.

Bibliography

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[2] H. Karl and A. Willig. A short survey of wireless sensor networks. Technical report, Telecommunication Networks Group, Technische Universitt Berlin, 2003.

[3] J. N. Al-Karaki and A. E. Kamal. Routing techniques in wireless sensor networks: a survey. *IEEE Wireless Communications*, 11(6):6–28, 2004.

[4] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 14–27, New York, NY, USA, 2003. ACM.

[5] Elizabeth M.Royer and Charles E.Perkins. An implementation study of the aodv routing protocol. In *Wireless Communications and Networking Conference (WCNC)*, number 4, 2000.

[6] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pages 90–100, Feb 1999.

[7] Jasmeet Chhabra. Tinyaodv implementation. TinyOS Source Code Repository, `http://cvs.sourcefourge.net/viewcvs.py/tinyos/tinyos-1.x/contrib/hsn/`.

[8] Nam N. Pham, Jon Youn, and Chulho Won. A comparison of wireless sensor network routing protocols on an experimental testbed. In *SUTC '06: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing - Vol 2 - Workshops*, pages 276–281, Washington, DC, USA, 2006. IEEE Computer Society.

[9] Brad Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, New York, NY, USA, 2000. ACM.

[10] İbrahim Körpeoğlu. Wireless communication fundamentals – part 1, 2008. Wireless Ad Hoc and Sensor Networks – Lecture slides.

[11] Abdalkarim Awad, Thorsten Frunzke, and Falko Dressler. Adaptive distance estimation and localization in wsn using rssi measures. In *DSD '07: Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, pages 471–478, Washington, DC, USA, 2007. IEEE Computer Society.

[12] Katelijne Vandenbussche. Fine-grained indoor localisation using wireless sensor networks. Master's thesis, Delft University of Technology, Faculty of Electrical Engineering, Mathematics, and Computer Science, 2005.

[13] Cesare Alippi and Giovanni Vanini. Wireless sensor networks and radio localization: A metrological analysis of the mica2 received signal strength indicator. In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 579–582, Washington, DC, USA, 2004. IEEE Computer Society.

[14] Chipcon Products from Texas Instruments. Cc1000 user manual. `http://www.ti.com/lit/gpn/cc1000`.

[15] Katja Schwieger and Gerhard Fettweis. Power and energy consumption for multi-hop protocols : A sensor network point of view. In *International Workshop in Wireless Ad-Hoc Networks*, 2005.

[16] Nana Adow Dankwa. An evaluation of transmit power levels for node localization on the mica2 sensor node. Technical report, Yale University, Electrical Engineering Department, 2004.

[17] Crossbow Products. Mica2 datasheet. `http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf`.

[18] Crossbow Products. Mica sensor board datasheet. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MTS_MDA_Datasheet.pdf`.

[19] Crossbow Products. Mib510 - serial interface board datasheet. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB510CA_Datasheet.pdf`.

[20] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, New York, NY, USA, 2003. ACM.

[21] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. Tinyos: An operating system for sensor networks.

[22] Philip Levis and Nelson Lee. Tossim: A simulator for tinyos networks. `http://www.cs.berkeley.edu/~pal/pubs/nido.pdf`.

[23] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107, New York, NY, USA, 2004. ACM.

[24] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the IEEE Infocom*, pages 1567–1576, New York, NY, USA, June 2002. USC/Information Sciences Institute, IEEE.

[25] Tijs van Dam and Koen Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180, New York, NY, USA, 2003. ACM.

[26] Crossbow Products. Mica sensor board user manual. `http://www.xbow.com/support/Support_pdf_files/MTS-MDA_Series_Users_Manual.pdf`.

[27] Multihop routing. `http://www.tinyos.net/tinyos-1.x/doc/multihop/multihop_routing.html`.

[28] Cc1000 radio stack manual. `http://www.tinyos.net/tinyos-1.x/doc/mica2radio/CC1000.html`.

[29] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, page 8020, Washington, DC, USA, 2000. IEEE Computer Society.

[30] James F.Kurose and Keith W.Ross. *Computer Networking, A Top-Down Approach Featuring the Internet*. Addison Wesley, 2004.

[31] Alper Rıfat Uluçınar. Tinyos and java sdk installation on linux platform. `http://wlab.cs.bilkent.edu.tr/labWiki/TinyOS/Howto_TinyOS_Debian`.

[32] Chartdirector, professional chart components for windows web applications. `http://www.advsofteng.com`.

Appendix A

User Manual of the E-Sense system

A.1 Installation

Please use the following steps for proper installation:

1. Install a linux operating system.
2. After installing linux, install *TinyOS 1.x* and *Java SDK* (detailed installation instructions of *TinyOS 1.x* and *Java SDK* can be found in [31]).
3. Install *Apache* server (any version), *PHP* server (any version) and *mySQL* server (version 5.0 or higher). The root folder of *PHP* pages must be under the "/var/www" directory.
4. Install *ChartDirector* library (used for constructing plots in PHP pages) according to the instructions on the webpage [32]. Make sure that the "lib" folder is under the "/var/www" directory.
5. Install "MySQL Connector J" to "/usr" folder. It can be downloaded from `http://www.mysql.com/products/connector/j/`.

6. Copy the "E-SenseMultihop" folder and its contents into the "/opt/tinyos-1.x/apps" directory.

7. Copy the "BerkRoute" folder and its contents into the "/opt/tinyos-1.x/apps" directory.

8. Copy the "BerkRouteJava" folder and its contents into the "/opt/tinyos-1.x/tools/java/net/tinyos" directory.

9. Copy the content of "PHPpages" folder into the "/var/www" directory.

10. Copy bash scripts "tinyos-1.x.sh", "serialset.sh", "mysqljava.sh", "config.sh", and "mysql.sql" to the "/opt" directory. (Note: If your PC has an embedded serial port, you need to delete the lines except the first line from "tinyos-1.x.sh" file. You should also change the specified "MySQL Connector J" path with yours in "serialset.sh" file.)

11. Copy the bash script "mysqlpassrm.sh" to "/opt/tinyos-1.x/tools/java/net/tinyos/BerkRouteJava" and "/var/www" directories. In each of these directories, execute that bash script by entering ". mysqlpassrm.sh ./" to the command line.

12. Under the "/opt/tinyos-1.x/apps" directory, create a file with the name "Makelocal". Enter the following text into this file and save it.
    DEFAULT_LOCAL_GROUP = 0x33
    PFLAGS += -DCC1K_DEF_FREQ=916700000

13. Now open a new bash terminal and access to root privileges.

14. Execute the mysql.sql script by typing "mysql < mysql.sql". This script creates database tables that are necessary for the system.

15. Execute ipchange.sh script in the directory "/var/www/DisplayData" by writing ". ipchange.sh *yourIP*". You must write the IP address of your PC instead of *yourIP*.

16. Execute the config.sh script. This script sets enviroment variables that are necessary for *TinyOS* to work. Now we must compile the program for sensor

motes and load compiled binaries into the flash memory of each sensor mote. For compiling the program, enter the command "make mica2" under the "/opt/tinyos-1.x/apps/E-SenseMultihop" directory. Before the program is loaded into the flash memory, the sensor mote must be connected to the PC via the serial port and the serial interface board (*MIB510*). Then type "make mica2 reinstall.ID MIB510=/dev/ttyUSB0" where ID corresponds to the sensor node ID. The ID of a node must be between 0 and 255. (If you have a serial port embedded on your PC, then replace "/tty/USB0" with "/ttyS0").

Please note that in the final configuration, the ID of the mote, which is connected to the PC, must be zero.

A.2 Execution and maintenance

Running the system only requires the following:

1. Open a new terminal and login as a root user.
2. Execute the config.sh script.
3. Enter the "/opt/tinyos-1.x/tools/java/net/tinyos/BerkRouteJava" directory and enter the command "java ProcessData" for execution.

Now the system should be working, receiving messages from sensor motes. Results and feedbacks can be observed from the E-Sense homepage `http://yourIPaddress` or `http://domainName`.

Maintenance is simple. Click the "admin login" link at the homepage. In the login page, enter the necessary username and password (by default, the username is "berk", password is "CER33CAH"). In the administrator control page, you can add the recently added nodes to the database. Deleting and modifying are also possible and easy to do.

For sensor motes which run out of batteries, you should simply change batteries with the fresh ones. There is nothing extra to do after this operation.