IMPLEMENTATION OF THE HEED CLUSTERING PROTOCOL WITH SLEEP SCHEDULING IN TINYOS 2 ON A WIRELESS SENSOR NETWORK TESTBED

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

CANAY ÖZKAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2013

Approval of the thesis:

IMPLEMENTATION OF THE HEED CLUSTERING PROTOCOL WITH SLEEP SCHEDULING IN TINYOS 2 ON A WIRELESS SENSOR NETWORK TESTBED

submitted by **CANAY ÖZKAN** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Gönül Turhan Sayan _____
Head of Department, **Electrical and Electronics Engineering**

Assoc. Prof. Dr. Elif Uysal Bıyıkoğlu _____
Supervisor, **Electrical and Electronics Engineering**

Examining Committee Members:

Prof. Dr. Semih Bilgen
Electrical and Electronics Engineering Department, METU _____

Assoc. Prof. Dr. Elif Uysal Bıyıkoğlu
Electrical and Electronics Engineering Department, METU _____

Assoc. Prof. Dr. İlkay Ulusoy
Electrical and Electronics Engineering Department, METU _____

Assoc. Prof. Dr. Cüneyt Bazlamaçcı
Electrical and Electronics Engineering Department, METU _____

Arda Akman, M.Sc.
Director, Argela Technologies, Inc _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: CANAY ÖZKAN

Signature :

ABSTRACT

IMPLEMENTATION OF THE HEED CLUSTERING PROTOCOL WITH SLEEP SCHEDULING IN TINYOS 2 ON A WIRELESS SENSOR NETWORK TESTBED

Özkan, Canay
M.S., Department of Electrical and Electronics Engineering
Supervisor : Assoc. Prof. Dr. Elif Uysal Bıyıkoğlu

September 2013, 65 pages

Energy efficient operation is often critical for wireless sensor networks (WSN), because network nodes operate on limited batteries. The aim of this thesis is to realize an implementation of an energy-efficient, scalable and competitive clustering protocol, HEED on MICAz motes using TinyOS 2 with further energy-efficiency enhancements. A working testbed is constructed, on which the network organizes itself to send periodic sensor measurements to a central station. In order to control overall energy consumption, power aware routing, topology control, time synchronization, sleep scheduling and transmision power adaptation problems are experimentally studied on the testbed.

Keywords: Wireless Sensor Networks, Energy Efficiency, Clustering, Sleep Scheduling, Time Synchronization

ÖZ

HEED KÜMELEME PROTOKOLÜNÜN UYKU PLANLAMASIYLA BİRLİKTE BİR KABLOSUZ ALGILAYICI AĞ SINAMA ORTAMINDA TINYOS 2 KULLANILARAK GERÇEKLEŞTİRİLMESİ

Özkan, Canay
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Elif Uysal Bıyıkoğlu

Eylül 2013 , 65 sayfa

Kablosuz algılayıcı ağlarda düğümler pil ile çalıştırıldığından enerjileri sınırlıdır. Bu yüzden enerji verimliliği çok önemlidir. Bu tez çalışmasının amacı, enerji verimli, ölçeklenebilir ve başkalarıyla rekabet edebilir bir kümeleme protokolü olan HEED'in bir uygulamasını, enerji verimliliğini daha da geliştirecek şekilde TinyOS 2 kullanarak MICAz algılayıcı düğümleri üzerinde gerçekleştirebilmektir. Algılayıcı ölçümlerinin merkezi bir istasyona periyodik olarak gönderilmesini kendini örgütleyerek sağlayan bir ağ sınama ortamı çalıştırılmıştır. Ağdaki enerji tüketimini kontrol edebilmek için; güç bilinçli yol atanması, topoloji denetimi, uyuma planlaması ve iletim gücü uyarlanması problemleri sınama ortamı üzerinde deneysel olarak çalışılmıştır.

Anahtar Kelimeler: Kablosuz Algılayıcı Ağlar, Enerji Verimliliği, Kümeleme, Uyuma planlaması, Zaman Senkronizasyonu

To my family

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to Assoc. Prof. Dr. Elif Uysal Bıyıkoğlu for her support throughout this study. Thanks to her, I had the freedom to explore on my own while getting benefited from the laboratory instruments. Also I would like to thank Sejjad Baghaee for giving his time to conduct experiments with me in the laboratory.

I would like to thank the Scientific and Technological Research Council of Turkey (TÜBİTAK) for providing the financial aids, since the wireless sensor network testbed used in this study was constructed with funds from TÜBİTAK Project 110E252.

I am very grateful to my family for their concern about my education throughout my life and their belief in me. Without their support I would not have been able to finalize this study.

Furthermore, I would like to thank to my all colleagues, but especially to: Onur Eker for showing great patience while listening to me and sharing his valuable thoughts about this study; Burcu Yargıçoğlu for pushing me -with her observations- to work without a break; Emine Çimen Öztürk, Selami Çiftçi and Onur Altunbaş for struggling to reduce my workload as the thesis jury date approached and lastly my manager at work, Arda Akman for being tolerant and trustful.

Finally, my special thanks go to my friends and also my respected elders: Mehmet Serdar Özcan and Onur Gültekin whose help were invaluable and to whom I am deeply indebted; Umut Deniz, Fatih Aksel, Eda Sezen Ata, Devrim Tokcan, Hakan Tuna, Sezgin Özyıldırım and Umut Özen who endured, motivated and supported me a lot from the beginning of this study to the end.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ADC	Analog-to-digital converter
AES-128	Advanced encryption standard with 128 bit key
AM	Active message
API	Application programming interface
CDMA	Code division multiple access
CSMA/CA	Carrier sense multiple access with collision avoidance
EECS	Energy efficient clustering scheme
EEUC	Energy efficient uneven clustering
EEPROM	Electrically erasable programmable read-only memory
FIFO	First-in, first-out
FTSP	Flooding time synchronization protocol
GUI	Graphical user interface
HAA	Hardware abstraction architecture
HEED	Hybrid energy-efficient distributed
iHEED	Integrated hybrid energy-efficient distributed
ID	Identity
ISP	In-system processor
LEACH	Low-energy adaptive clustering hierarchy
MAC	Media access control
MEMS	Micro-electro-mechanical systems
MoteIF	Mote interface
OS	Operating system
P_c	Initial intra-cluster power level
PC	Personal computer
PEGASIS	Power efficient gathering in sensor information system
P_t	Initial inter-cluster power level
RAM	Random-access memory
RBS	Reference broadcast synchronization
R_c	Initial intra-cluster transmission range
RF	Radio frequency
RSSI	Received signal strength indicator
R_t	Initial inter-cluster transmission range

RX	Receive
SNR	Signal-to-noise ratio
TCP	Transmission control protocol
TDMA	Time division multiple access
TPSN	Timing-sync protocol for sensor networks
$T_{routing}$	Period of routing packet broadcasts
T_{sync}	Period of time synchronization packet broadcasts
TX	Transmit
USB	Universal serial bus
VLSI	Very-large-scale integration
WSN	Wireless sensor network

CHAPTER 1

INTRODUCTION

Recent advances in electrical engineering and computer science, especially in micro-electro-mechanical systems (MEMS), very-large-scale integration (VLSI) and operating system theory, have made wireless sensor network (WSN) technologies an important and promising research area. As the circuitry became smaller and cheaper, the idea of cooperatively sensing physical or environmental conditions, such as magnetic field intensity, temperature, light intensity, sound, vibration, pressure, motion or pollutants with the help of a network of sensor nodes has become feasible.

A typical WSN communicates in a multi-hop fashion and aims to report an environmental event to the end users through a gateway as shown in Figure 1.1.

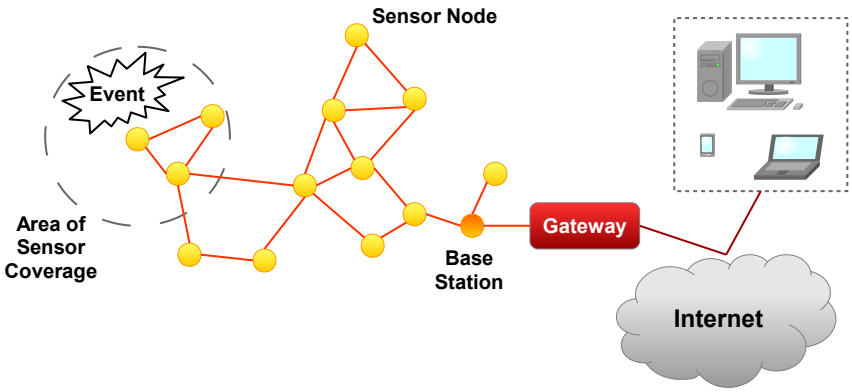

Figure 1.1: A typical WSN

A sensor node, which is the smallest part of a wireless sensor network, has three main capabilities: sensing, processing and communicating. The components required for the proper operation of a sensor node are shown in Figure 1.2. The sensor circuitry is responsible for observing the physical parameters of the environment and transforming them into electrical signals. The controller is responsible for processing these electrical signals and controlling all the components of the sensor node. The communication device is responsible for sending and receiving data in a wireless fashion. Furthermore, a sensor node has a limited power supply and a limited memory, where received or processed data can be stored. Due to its limited power supply, which typically is a

decaying battery, a sensor node must consume minimum energy. Due to its limited memory, a sensor node must hold and process data as seldom as possible. At this point, the necessity of an energy-aware algorithm for a scalable wireless sensor network arises. Within the literature many propositions were developed in order to achieve energy effi-

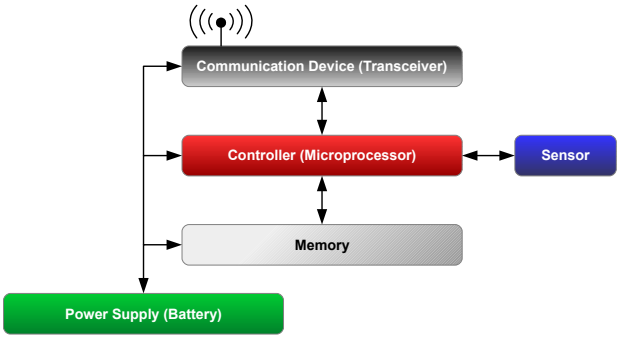

Figure 1.2: Components of a sensor node

ciency. These propositions involve duty cycling, energy efficient scheduling, scheduled rendezvous, on demand wakeup scheme, energy efficiency through directional antennae, clustering, energy efficient routing, energy efficiency through topology control, data aggregation and remote power supply [1][2]. In order to reach the energy efficiency goal, a subset of these propositions: sleep scheduling, clustering and transmission power adaptation have been applied in the present study.

- **Sleep scheduling** may help energy efficiency by making use of the sleeping modes of the sensor nodes. However, sleep scheduling mechanisms have highly application dependent nature. Sensors used, network structures, deployment of the nodes, sensing areas, transmission ranges, detection models and time synchronization between sensor nodes are some factors that effect the overall design of these mechanisms [1].

- **Clustering** can not be separated from routing and topology control techniques when a WSN architecture is considered. Decisions made by applying a clustering algorithm effects the resulting topology and produces different routes for the packets headed towards the base station (see Figure 1.3). Clustering designed in accordance with routing and topology control is valuable since it brings the wireless sensor network the advantages of scalability, longer lifetime, distributed control, reuse of resources, reduction of the amount of information propagation and reduction of the amount of state information storage compared to other routing schemes in WSNs [3].

- **Transmission power adaptation** may have several benefits in a wireless communication link by means of both decreasing and increasing transmission power levels. On one side, a power increasing policy provides reception with a higher signal-to-noise ratio (SNR) which reduces bit error rates and leads to reduced packet retransmissions. On the other side, a power decreasing policy provides less energy consumption per packet and less interference. Hence, a wise and application-specific power adaptation scheme should be applied in a WSN in

order to reduce the overall energy consumption and the interference problems which may end up with network malfunctions.

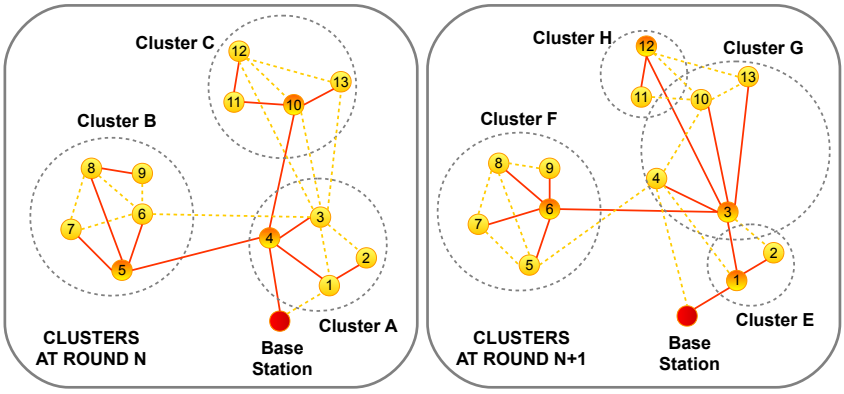

Figure 1.3: Clustered architecture

In this thesis work, the hybrid energy-efficient distributed (HEED) clustering protocol with energy efficiency enhancements through sleep scheduling and transmission power adaptation in a wireless network of accelerometer sensors is implemented. To the best of our knowledge, this work presents the first WSN implementation of the HEED clustering protocol in TinyOS 2 with transmission power adaptation, time synchronization and sleep scheduling all together. In order to validate the synchronized operation of the sensor nodes and observe the effect of the applied transmission power adaptation algorithm and sleep scheduling scheme on the overall energy consumption of the network, performance of the implementation is evaluated experimentally on a WSN testbed composed of MICAz motes.

The organization of the rest of this document is as follows:
In Chapter 2 Related Work
In Chapter 3 Preliminaries
In Chapter 4 Implementation
In Chapter 5 Experimental Results
In Chapter 6 Conclusion and Future Work

CHAPTER 2

RELATED WORK

2.1 Clustering

In hierarchical clustering approach, there are many clusters, which are composed of comparably small number of sensor nodes. All sensor nodes are to play one of the two possible roles in a cluster. First one is the cluster head role. Basically a cluster head is the node which is responsible for collecting all the sensor data in that cluster and passing it to the other cluster heads or to the base station. It may also control regular nodes' activities, transfer the disseminated packets to regular nodes, organize the medium access in the cluster or process the received data depending on the protocol design. The second one is the regular node role. A regular node is only responsible for sensing the conditions in its coverage and reporting them to the cluster head either in a periodical way or in request. A cluster head can forward the data to larger distances in order to be able to reach other cluster heads and reduce the latency in the network. However, the hierarchical multi-hopping behavior between the cluster heads towards the base station becomes a disadvantage as the distance between cluster heads increase, since it is a known fact that energy consumption increases with a bigger proportion than the square of the distance. Within the past decade, there have been many studies which investigated the hierarchical clustering approach and the energy consumption issue in WSNs. Low-energy Adaptive Clustering Hierarchy (LEACH), Power Efficient Gathering in Sensor Information System (PEGASIS), Energy Efficient Clustering Scheme (EECS), Energy Efficient Uneven Clustering (EEUC) and HEED are the most significant protocols proposed in those studies.

LEACH, among the earliest and most popular hierarchical cluster-based protocol proposals, is worthy of discussion. It assumes that all sensor nodes in the network is able to transmit with enough power to reach the base station if needed. It also assumes that the nodes can modify their transmission powers and have enough computational capacity for proper operation [4]. In LEACH, the sensor nodes locally organize themselves as cluster heads and cluster member nodes. Selection of sensor nodes as clustering head in a rotating manner is the main objective. Rounds are the major parts of the operation of the LEACH. There are two phases in each round: the set-up phase and the steady-state phase. In the setup phase, organization of the clusters is performed. Each node's decision to become a cluster head or not is made during this phase. This decision depends on the suggested density of the cluster head nodes in the network and the number of times being a cluster head so far. If a node is elected as a cluster head, it sends a message to the other nodes to indicate itself as a cluster head. The other

nodes receive the message from the cluster head and depending on the signal strength, they will send a message to join the cluster. In the steady state phase, delivery of the data to the base station is performed.

For even energy distribution among sensor nodes, cluster heads change at each round. A cluster head receives data from the sensor nodes which are members of its cluster. Then, if data aggregation is applied, it compresses the received data before sending to the base station directly. Furthermore, inter cluster and intra cluster collisions are diminished by using Time Division Multiple Access (TDMA) or Code Division Multiple Access (CDMA).

However, LEACH has some disadvantages [5]. It performs single-hop inter-cluster communication, directly from cluster heads to the base station, which is not applicable to networks deployed on a wide area. Although cluster head rotation is performed at each round for load balancing, LEACH cannot ensure a real load balance for sensor nodes with different amounts of initial energy, since it does not consider nodes' remaining energy during its setup phase.

PEGASIS, which is an improvement of LEACH, has the main idea that communication is only performed within close neighbors and each node becomes a leader in turns for transmission of data packets to the base station [6]. Nodes can be arbitrarily positioned and it is assumed that positioning knowledge is a feature of all sensor nodes.

In PEGASIS, the base station may assign a chain and then broadcast it to all nodes or nodes may locally use a greedy algorithm to form a chain. The furthest node from the sink is the starting point of the chain, the next node of the chain is the closest node to it. If one of the nodes in the chain die, a new chain is reconstructed by commencing again from the furthest node from the sink. In each round, nodes transmit their own data to their closest neighbors, neighbor nodes receive the data and merge the data with their own data. The merged data is sent from node to node and finally the leader nodes send the merged data to the base station. Data transmission from the ends of the chain is started by the help of a control token passing approach, which is initiated by the leader node.

PEGASIS outperforms LEACH for different networks sizes and topologies by providing uniform dispersion of the energy load in the network. However, it has some disadvantages [5]. One disadvantage emerges while constructing the chain. Network topology is needed to be completely viewed by all nodes and they must be capable of communicating directly with the sink. Another disadvantage is that complete database of the location of all other nodes must be maintained by all nodes. Requirement for the global knowledge of the network and employment of the greedy algorithm make PEGASIS a non-scalable protocol.

EECS, which is proposed in [7], is a clustering algorithm used for periodical data gathering. Similar to LEACH, the network is partitioned into several clusters and single-hop communication between the cluster heads and the base station is performed. Cluster head candidates try to reach to cluster head level for a round. They broadcast their residual energy to neighbor nodes. Receivers of these residual energy messages seek for nodes having more residual energy than themselves, if they do not find any, they become cluster heads. EECS extends LEACH by dynamic sizing of clusters based on cluster distance from the base station. A weighted cost function, which takes the

node residual energy and workload into consideration, is introduced for a regular node to make the decision of being cluster head.

EECS balances intra-cluster energy consumption and inter-cluster communication load by taking into account energy and distance. In addition, EECS provides low message overheads and uniform distribution of cluster heads compared to LEACH [5]. However, there exist some trade offs. Clusters with a larger distance to the base station need more energy for transmission of data packets when compared to those with a shorter distance. Also nodes need to know globally about the distances between the cluster heads and the base station. Besides, EECS produces much more control overhead complexity because of the fact that all nodes must compete to become cluster heads.

EEUC is a distributed clustering algorithm proposed in [8]. Unlike LEACH, in EEUC, localized competition elects the cluster heads. The unequal clustering approach to balance the energy consumption among the cluster heads and solve the hot spot problems makes EEUC competitive.

In EEUC, for cluster head election, random numbers are generated by the nodes. If a node's generated number is greater than a threshold, this node is activated for cluster head election by broadcasting a compete message within a competition radius, which is smaller as the node gets close to the base station. Thus, if a cluster is close to the base station, it has a comparably small cluster size. This decreases power consumption during the intra-cluster data processing and saves more energy for the inter-cluster relay traffic. If a sensor node is to be a cluster head, first its residual energy competes with the other nodes' residual energy. If the sensor node succeeds to become a temporary cluster head, then it competes with the other temporary cluster heads in order to become a real cluster head.

The most significant disadvantage of EEUC is that it actually does not balance the energy consumption among cluster heads, since an unequal clustering mechanism is used in EEUC to solve the hot spot problem.

HEED proposed in [9] is a clustering protocol which considers energy-efficiency explicitly. HEED uses two parameters for periodically electing a cluster head and constructing a cluster. One parameter is node residual energy. The other parameter is intra-cluster communication cost. With the usage of first parameter, cluster heads often have relatively higher residual energy compared to regular nodes. With the usage of the second parameter, regular nodes join the cluster of the cluster heads with the least communication cost. If no cluster head announcement is received by a node during the iterations in a cluster formation round, it announces itself as a cluster head. After the cluster formation round, HEED allows the cluster heads to send the data to the base station in a multi-hop fashion with the help of a routing protocol.

HEED has many advantages. It produces well-distributed cluster heads for load balancing and compact clusters in the network. It minimizes control overheads since it is a fully distributed algorithm. It provides fair distribution of the energy consumption among the sensor nodes. Additionally, necessity of communicating in a multi-hop fashion between cluster heads and the base station supports scalability.

However, HEED also has several limitations. One important limitation is that more cluster heads may be generated than the expected number after a cluster formation

round and this accounts for an unbalanced energy consumption in the network. Another limitation is that some cluster heads, especially near the sink, may die earlier, since hot spot problem also exists for HEED [10]. Knowing its limitations and the fact that EEUC performs better in load balancing and energy efficiency, HEED is intentionally chosen for the formation of the clusters in the implementation. This is because of the complexity of EEUC which is higher than the complexity of HEED. Besides, EEUC does not solve the hot spot problem either [5].

HEED is explained in detail in Chapter 4 Section 4.2.

2.2 Time Synchronization

Reference Broadcast Synchronization (RBS) is unlike the conventional time synchronization protocols where the receiver synchronizes to the timestamp broadcast by the sender. In RBS receivers synchronize between themselves thanks to a beacon transmitted by a third party. The beacon is a simple signal without any time information. It helps the receivers to compare their clocks and find out the phase offset between them. The beacon is transmitted to two receivers and those receivers store the receiving time based on their own clocks. This is sufficient for them to compare their receiving times to obtain a time offset. In a larger configuration with more than two receivers, more broadcasts may be needed, since the more the broadcasts are, the better the precision of the synchronization is [11].

In RBS, there is no need for a root sender in synchronization, which comes in very handy by removing the sender uncertainty. Also the propagation time uncertainty can be omitted, since the beacon signal is considered to reach all the receivers at the same time in small-transmission-range networks. However, omitting the propagation delays makes RBS unsuitable for large networks. Additionally, in large networks more beacons must be broadcast, which is a drawback along with the receiver uncertainty problem faced in RBS [12].

Timing-sync Protocol for Sensor Networks (TPSN), where the synchronization is again performed using the sender-receiver logic, constructs a tree to organize the network topology. In the first stage of the synchronization, which is called the level discovery phase, nodes are separated into levels such that the only node in level 0 is the root node. In the second stage, the synchronization phase, nodes at level i synchronize with the nodes at level i-1, which helps them to synchronize with the root node at the end [13].

In the level discovery phase, which is performed across the network, a root node, which the other nodes can replace at certain intervals, is chosen. The root node starts the level discovery by broadcasting the level discovery packet, which contains its identity and level. The neighbors, which receive the packet, become part of level 1 and broadcast a level discovery packet to their neighbors. This will repeat until all nodes are included in a level. After all nodes are included a level, the tree topology formation ends with the root node at level 0 and the other nodes at the higher levels.

The synchronization phase is based on a two-way sender to receiver communication. As in the level discovery phase, the synchronization phase starts from the root node

and goes on across the network. The root node starts the synchronization phase by sending a time synchronization packet to level 1 nodes. Level 1 nodes wait for a while and start the two-way messaging with the root node. Then, the root node sends the acknowledgement packet and level 1 nodes synchronize with the root node. Level 2 nodes can hear level 1 nodes' as they are neighbors. Thus, they wait for a while and then start the two-way messaging with level 1 nodes. This procedure goes on until all nodes are synchronized with the root node.

Since it is a multi-hop protocol, transmission range is not a problem in TPSN [12]. But four delays are in question: sending time, access time, propagation time and receiving time. TPSN does not remove them. Therefore, it does not remove the sender uncertainty, unlike RBS, but it reduces its negative effects. It tries to diminish it using low level time stamping packets in the media access control (MAC) layer. It is supposed to affect the total synchronization error at a very small rate. This approach is said to make the precision of TPSN two times better than that of RBS, which makes sender-receiver synchronization better than receiver-receiver synchronization [13]. Besides, with TPSN, the tree structure allows the timing information to be transferred properly across the network, which makes it a better design for multi-hop networks. However, it requires a new level discovery phase when the root node or the network topology changes, which causes additional traffic in the network.

Flooding Time Synchronization Protocol (FTSP) is a sender-receiver time synchronization protocol with resemblance to TPSN for having one root node to which all other nodes are synchronized, but it tries to correct some drawbacks of TPSN.

In FTSP, the root node broadcasts the global time information as its timestamp at the moment of transmission. Receivers, upon getting this information, become able to calculate their clock offsets by comparing their local times and the sender's timestamp. The packets broadcast are timestamped in the MAC layer at both the sender and the receiver ends. The precision of clock is achieved by making corrections with linear regression.

In order to get synchronized with the root node, nodes communicate amongst themselves to share the timing information in an ad hoc manner. Thus, the network topology for FTSP is mesh as distinct from TPSN, where a tree topology is used [14].

FTSP is resistant to node and link errors by making use of the flooding of synchronization packets. It makes FTSP also adaptable to inevitable topology changes, where a new root node must be assigned intermittently. Timestamping in the MAC layer helps to promote precision and lessen jitter, which leaves only the propagation time error as drawback [12]. To overcome that, multiple time stampings and linear regression are used to calculate and fix clock offsets.

PulseSync concerns about speeding up the process of distributing the clock information and lessening the number of messages in this process, i.e. by flooding only a pulse with a breadth-first search tree manner [15]. The pulse is broadcast into the network by a root node. Other nodes wait for this pulse from their neighbors. When they receive it, they forward it quickly to minimize the error introduced by themselves. As in FTSP, in order to compensate for and correct the effects of the jitter on the node clock, the linear regression method is used.

PulseSync is proven to be more accurate than FTSP for mid-sized sensor network applications where a relatively stable network topology condition is satisfied.

CHAPTER 3

PRELIMINARIES

This chapter introduces software and hardware components extensively used in the proposed WSN implementation. TinyOS operating system, MICAz motes, CC2420 radio chip, MTS310CB sensor board and MIB520CB programming board are described shortly.

3.1 TinyOS

TinyOS is an operating system specifically designed for running low-power wireless applications on sensor nodes (motes), which generally tend to have tiny, low-power microcontrollers. It provides a component model, which defines how you write small, reusable pieces of code and compose them into larger abstractions. Thanks to its concurrent execution model, which defines how components interleave their computations as well as how interrupt and non-interrupt code interact, actions of many components at the same time are supported while requiring little random access memory (RAM). It also provides a hardware abstraction architecture (HAA), which defines how to build up from low-level hardware to a hardware-independent abstraction [16]. Furthermore, it has many advantages over traditional operating systems for the targeted platforms:

- It provides a lightweight and ultra-low power run-time environment and requires low memory compared to other operating systems, since neither of kernel, process management, virtual memory, dynamic memory allocation and software exceptions exist in it.
- It has very aggressive mechanisms when the aim is to save power.
- Developers has easy access to application programming interfaces (APIs), services and abstractions, such as sensing, communication, storage, and timers, which are critical in WSN applications.
- Services and components can be reused regardless of worrying about unforeseen interactions.
- It runs on over a dozen generic platforms.
- It is continually evolving. Within the TinyOS community, a working group deals with issues within the operating system (OS) and improves existing services and adding new ones.

It must be stated upfront that all properties presented in this section are for the TinyOS version 2.x and all of TinyOS code, including itself is written in the nesC language, which is a C dialect with features to reduce RAM and code size with significant optimizations, and help prevent low-level bugs like race conditions [16].

Introductory description of some important concepts in TinyOS-2.x can be given as follows:

- An application in TinyOS, is a nesC program that is built up by several components. There are two kinds of components: modules and configurations. A module consists of nesC code, which implements of one or more interfaces by declaring variables and calling functions. Configuration consist of nesC wiring code, which connects components together.
- A component can be composed of wirings, frame, interfaces, events, commands and tasks.
- A frame is the storage related to a component. It represents the static memory allocations done for the variables of the component at compile time.
- TinyOS does not encourage dynamic memory allocation.
- Interfaces are the list of exposed events and commands. Interactions between components are specified by interfaces: the interface user makes requests (or calls commands) on the interface provider, the interface provider makes callbacks (or signals events) to the interface user. Calling a command or signaling an event is just a function call.
- TinyOS has an event-driven architecture. Events can preempt tasks, signal events, call commands or post tasks. In general, execution is driven solely by timer and radio events.
- Interrupts are naturally asynchronous. They trigger lowest level events, stop the current execution and start running preemptively.
- Tasks perform the background computation in a component. They are simply functions of a component which are scheduled to run later, rather than now. TinyOS guarantees that posted tasks eventually run. Furthermore, tasks can not preempt each other. However, a task can be preempted by a hardware interrupt.
- The TinyOS task scheduler has a fixed-length first-in, first-out (FIFO) queue.
- There is no kernel/user space differentiation in TinyOS.
- Time consuming operations are split-phase in TinyOS. A command call to start lengthy operation returns immediately. The called component signals an event when the operation is complete. As a result, TinyOS only needs a single stack and all components in the application use this stack.
- Networking (e.g. powering the radio on/off or sending a packet) and sensor sampling operations (i.e. reading analog-to-digital converter (ADC) output of the sensor) are generally split-phase since they take a long time for hardware operations to complete.

- Each node has a unique 16 bit address, am_addr_t, which is specified by the user while installing the binary to the motes.
- There are several address constants, but one of them is special: TOS_BCAST_ADDR (0xFFFF). Because it is reserved for broadcasts.
- The lowest networking layer exposed in TinyOS is called active messages. Typically, active message (AM) is provides unreliable, single-hop packet transmission and reception. Each packet is identified by an AM type, an 8-bit integer that identifies the packet type. AM type is like transmission control protocol (TCP) ports. They are used to automatically dispatch received packets to an appropriate handler.
- In TinyOS, the standard message buffer is message_t (see Figure 3.1). message_t has header, data and meta-data fields. A packet holds a single AM packet type in its header. Also packet length is also stored in the header. Data field is reserved for user payload at a fixed offset. Its size in bytes, TOSH_DATA_LENGTH is specified by the user at compile time [17]. The metadata field is reserved for per packet data, such as RSSI and timestamp fields which are not transmitted. Radio stack uses or collects this fields' data.

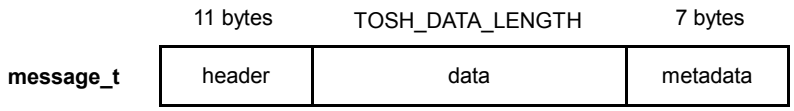

Figure 3.1: Structure of the message_t

- TinyOS provides timer interfaces which are used for measuring time and triggering events at specific times. They typically use 32-bit numbers, which wrap around in about 2 days, to represent time.
- TinyOS applications can also send or receive data over universal serial bus (USB) connection to an attached personal computer (PC). The net.tinyos.message Java package in TinyOS support directories provides a Java class, named mote interface (MoteIF), which makes it possible to transmit and receive messages from a mote to a PC or vice versa.

Features of TinyOS and nesC are extensively used in the proposed energy-efficient WSN implementation. However, the details (such as nesC syntax, programming the motes, debugging, etc.) are out of the scope of this thesis. Please refer to [16], [18] and [19] for more information.

3.2 MICAz motes

The Crossbow MICAz motes are used for the proposed WSN implementation (see Figure 3.2). It is a 2.4 GHz Mote module used for enabling low-power WSNs [20].

Figure 3.2: MICAz mote

Some important features of the MICAz mote are:

- Atmel ATmega128L low-power microcontroller
- CC2420 Radio Chip
- 128K bytes Program Flash Memory
- 10 bit 8 channel ADC
- The 51-pin expansion connector
- 75m to 100m Outdoor Range
- 20m to 30m Indoor Range
- Powered by 2 x AA batteries

Please refer to [20] for more details.

3.3 CC2420 Radio Chip

The Chipcon CC2420 is the radio frequency (RF) transceiver on the MICAz motes. It is specifically designed for low-power and low-voltage wireless applications [21]. Its key features are:

- IEEE 802.15.4 compliant RF transceiver
- 2400 MHz to 2483.5 MHz ISM band, programmable in 1 MHz steps
- Direct sequence spread spectrum radio which is resistant to RF interference and provides inherent data security
- 250 kbps Transmit (TX) data rate
- Programmable output power
- 128 Receive (RX) + 128 Transmit (TX) bytes data buffering
- Hardware MAC encryption (Advanced encryption standard with 128 bit key (AES-128))

3.4 MTS310CB Sensor Board

The crossbow MTS310CB is a sensor board for the MICAz mote (see Figure 3.3). It can be connected to the MICAz via the standard 51-pin expansion connector. It has a 2-axis magnetometer, 2-axis accelerometer, microphone, sounder, light and temperature sensors. In the implementation, sensor data is provided by the accelerometer of this sensor board.

Figure 3.3: MTS310CB sensor board

Please refer to [22] for more details.

3.5 MIB520CB Programming Board

The Crossbow MIB520CB is a user interface board, which provides USB connectivity to the MICA family of motes for communication and in-system programming. It has an on-board in-system processor (ISP) to program the motes. Having TinyOS installed in the host PC , code is downloaded to the ISP through the USB port and ISP programs the code into the mote.

Please refer to [20] and [23] for more details.

CHAPTER 4

IMPLEMENTATION

In Figure 4.1 a block diagram of the components, which visualize the architecture of present implementation of HEED clustering in a single sensor node together with other features, can be seen. The following sections explain why Multi-hop Routing, HEED Clustering, Transmission Power Adaptation, Time Synchronization, Radio Sleep and Event Scheduling, Accelerometer Data Sampling and Forwarding, Energy Consumption Calculation components are required and how they operate in the implementation. Additionally, the interaction between these components is described in detail.

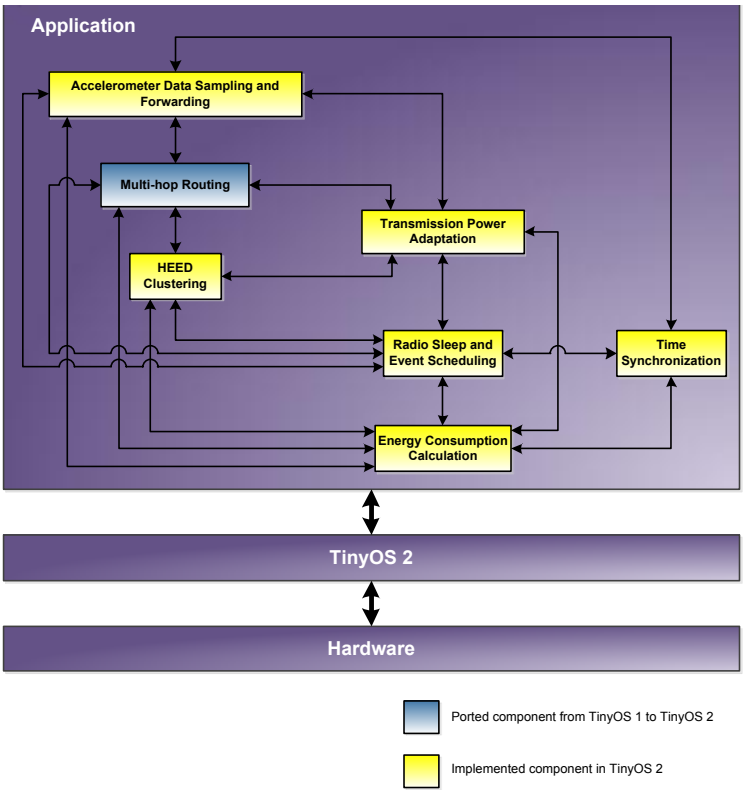

Figure 4.1: Architecture of the presented implementation in a single sensor node

Please note that sleep scheduling for the microcontroller of MICAZ is not an issue in this implementation, since the energy consumption of the microcontroller is automat-

ically controlled by TinyOS in such a way that microcontroller operates at the lowest possible power state [16].

4.1 Multi-hop Routing

Wireless ad-hoc networks are naturally very different from networks powered by electrical wiring. They are characteristically not reliable over the long term. Nodes may suddenly stop working or may become unreachable. Multi-hop routing protocols are needed to handle these dynamic changes in network topology. In addition, broadcasting, which is energy and time wasting, is prevented by dynamically determining the best parent node for a transmitting node.

Multi-hop routing protocols, however, bring a limitation to aggregate data rates [24]. Applications should maintain average message frequency below a rate, since higher rates can lead to congestion and overflow of the communication queues.

The TinyOS 1 release includes a library that provide an ad-hoc multi-hop routing protocol above the physical layer and MAC layer. According to this protocol all nodes have a single final destination node which is the base station. To reach it, they continuously apply three processes which constitute the skeleton of this protocol: link quality estimation, neighbor table management and route selection [25][26]. Each node has a two-way link estimator to determine the quality of the link with the neighbor nodes. Here, neighbor table management process is needed to select the neighbors to be put in the table, since there exist a severe memory limitation on the motes. These two processes together provide the possible connections between nodes to the route selection process. Then, the route selection process constructs a feasible spanning tree topology which is a subgraph of the logical connectivity graph. Thanks to these processes, the network can maintain a healthy and reasonably reliable communication.

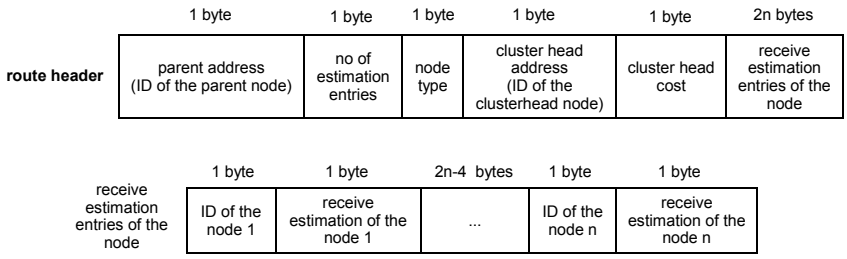

Figure 4.2: Structure of the routing packet

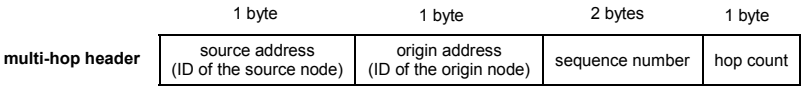

Figure 4.3: Structure of the multi-hop header

There are three components which handle this multi-hop routing protocol in TinyOS 1. First one, MultiHopRouter software component, is responsible for encapsulating the other two. Second one, MultiHopLEPSM component is responsible for broadcasting distance to the base station and link quality estimates, holding and managing the neighbors' information, updating the fields of the multi-hop header, selecting parents, eliminating duplicates and avoiding cycles. The routing packet periodically broadcast by MultiHopLEPSM component has 6 fields: parent address, number of estimation entries, node type, cluster head address, cluster head cost and receive estimation entries of the node respectively (see Figure 4.2). Cluster head address and cost fields are embodied in this packet for convenience and performance issues. The receive estimation entries field contains identities (IDs) and receive estimations that belong to a group of neighbor nodes. Third one, MultiHopEngineM acts as an interface for route selection and is mainly responsible for forwarding and sending/queuing of multi-hop data packets. The multi-hop header is inserted in front of all data and routing packets to be sent by MultihopEngineM. The header contains source address, origin address, sequence number and hop count fields (see Figure 4.3). All together, these components realize automatic transmission of the link quality estimates, selection of the optimal route and forwarding the multi-hop data traffic.

Figure 4.4 shows the flow of data through the processes which constitute this multi-hop routing protocol. The details of realization of these processes are explained below.

Parent selection is based on a distance-vector based algorithm, where number of hops is used as the measure of distance. At startup, number of hops information is initialized to ROUTE_INVALID (a value that indicates the route is not valid) for all nodes. The neighbor nodes of the base station are capable of maintaining a single-hop link with it, they select base station as parent and set their hop count 1. They are the first nodes to set their hop count to a value other than ROUTE_INVALID. Hop count information disseminates towards the network through these nodes. At the moment of parent selection, each node sets its current hop count as:

$$C_{H_{node}} = C_{H_{parent}} + 1 \tag{4.1}$$

where $C_{H_{node}}$ is the current hop count of the node and $C_{H_{parent}}$ is the hop count of the parent node. Parent selection procedure for the neighbor nodes of the base station is quite easy. Since they are already neighbor to the sink node. However, for the rest of the nodes parent selection procedure is different: routing algorithm sorts the neighbor table in the order of hop counts and extracts a set of potential parents which have the minimum hop counts. From the nodes in this set, it selects the node with the least link cost as parent. The algorithm also checks the link quality estimates, cycles and sink reachability in the selection procedure. If the **link quality estimates** drop below a threshold, a cycle is detected or the sink becomes unreachable, the node switches to a new parent. The link quality between two neighbor nodes is determined with the send and receive estimations. A node's average receive estimation, $RE_{average}$, from a neighbor node is calculated using the exponentially weighted moving average principle as:

$$RE_{average} = \frac{RE_{old} + 3 \times RE_{new}}{4} \tag{4.2}$$

where RE_{old} is the previous average receive estimation and RE_{new} is the latest calcu-

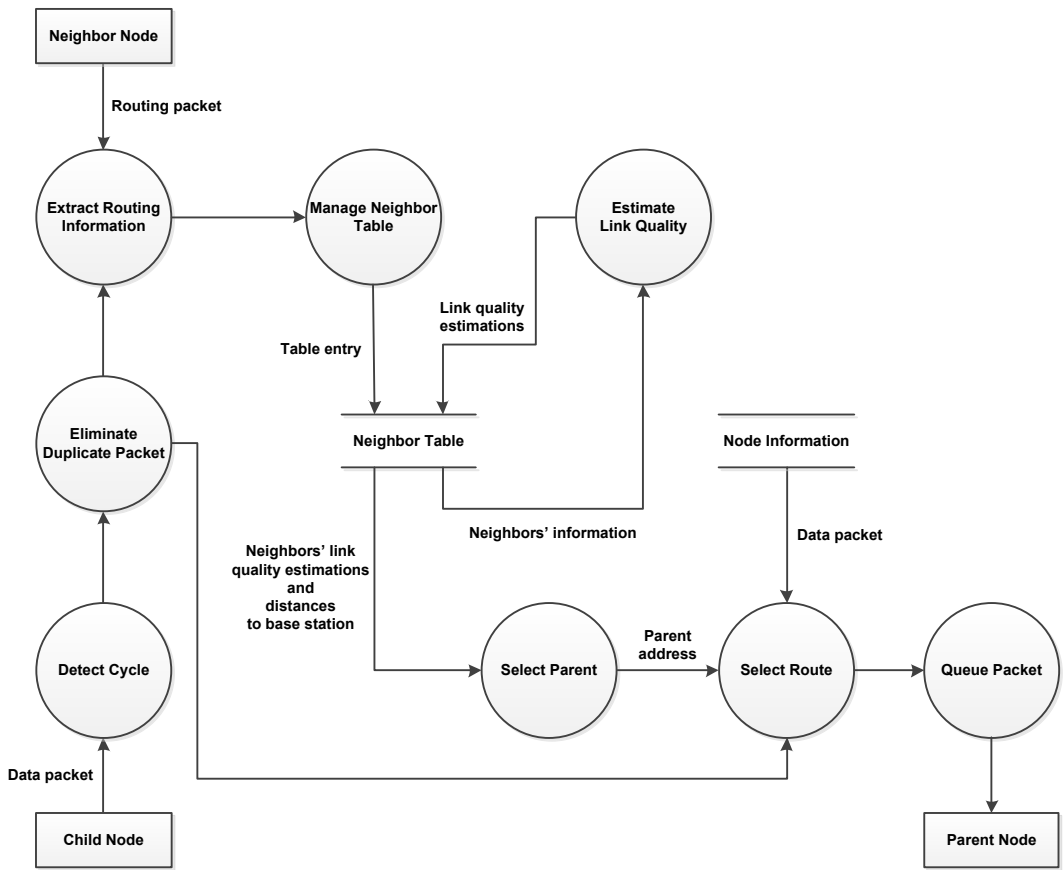

Figure 4.4: Multi-hop routing protocol data flow diagram

lated receive estimation:

$$RE_{new} = \frac{255 \times C_{P_{received}}}{C_{P_{received}} + C_{P_{missed}}} \tag{4.3}$$

where $C_{P_{received}}$ is the number of packets received and $C_{P_{missed}}$ is the number of packets missed from the last update up to now. A node's send estimation to a neighbor node is the receive estimation of the neighbor node which belongs to the originating node. This estimation is sent to the node with the help of routing packets. If the node does not receive a minimum number of routing packets for some time, send estimation is updated as:

$$SE = \frac{SE_{old}}{2} \tag{4.4}$$

where SE_{old} is the previous send estimation received from the neighbor node. Nodes can not be parents, if they have estimations below a minimum threshold, where low values indicate a poor quality link. Also nodes can not be parents if their two way link cost to the supposed node is above a maximum threshold, where link cost, LC, is estimated as:

$$LC = \frac{LC_{max}}{SE \times RE_{average}} \tag{4.5}$$

where LC_{max} is a constant value that link cost can have at maximum. The network should be responsive enough to the dynamic features of a WSN. For instance, say node A's parent is node B. For some reason the link between A and B gets broken. After a while, A should replace B with another neighbor since link cost becomes relatively high. To make this possible, described link estimation algorithms are run with a predetermined period and parent selection is done just before sending a multi-hop data packet. If A has no other potential parents available, then it declares itself to have no parent, disjoins from the tree, and sets its hop count to ROUTE_INVALID. To minimize the packet drops, considering link quality estimates in this kind of a mechanism is a must. Before a routing packet is broadcast, send estimates from the neighbor table are sorted and only the best estimates are included in the packet. This way poor links can be filtered out from the route construction process. Also applying shortest path routing (minimum number of hops) algorithm only to links whose estimated link qualities are above a predetermined threshold increases the depth of the network. This may result in a longer path with fewer packet drops. However care must be taken, since cell density and physical deployment may result in a connectivity graph where the set of nodes whose link quality estimations are above threshold may fail to connect the network [25].

Duplicate packet elimination is an important part of this multi-hop routing protocol. Before sending a multi-hop data packet, the routing layer at the originating node appends the sender ID and a sequence number in the multi-hop routing header. When a data packet received, the receiving parent node finds the latest sequence number for that node from the neighbor table. If the parent finds another incoming packet with the same sender ID and originating sequence number, it means that this packet is duplicated and it will not be processed or transmitted to the other nodes.

In a tree network topology, the nodes with smaller depths must be able to forward more packets. **Queue implementation** requirement should be fulfilled to ensure that messages are not dramatically dropped at these nodes due to congestion. Queues of the messages to be forwarded and messages originated are combined into a single FIFO transmission queue under the assumption that data rates are low and each node's data has the same priority in the network. Therefore, if there is congestion at a receiving node, the originated packets have the same chance to be forwarded as the packets to be forwarded.

Route selection algorithm for a received or an originated data packet is as follows: If a node does not have a parent and is originating a packet, it broadcasts the packet for seeding the network. If the node does not have a parent and is not originating the packet, route selection fails. If the node has a parent and is just forwarding the packet, it checks whether the packet is cycled or not. If it is cycled, again route selection fails. If not, it checks whether it is a duplicate packet or not. If it is a duplicate, again route selection fails. If not, the packet is sent to the previously selected parent.

Since number of hops information is used in the parent selection procedure, **cycles** are not likely to occur. However, in the route selection procedure, because of broadcasting some packets for seeding the network, cycles may occur. Besides, the packets may return to the originating node at the transition periods of new parent selection. Thus, cycles are simply detected by evaluating the sequence number field in the multi-hop packet header. If a packet having an equal or less sequence number is to be forwarded,

the forwarder node drops the packet. This is a light weight mechanism which avoids loop formation and breaks cycles.

Neighbor table holds the most recent entries for all nodes in order to make the described link estimation parent selection mechanism work accurately (see Table 4.1). Therefore, **the management of the neighbor table** plays a crucial role in an ad-hoc and self-configuring WSN. It has three main functionalities: insertion, eviction and reinforcement [25]. Each coming routing packet is analyzed, the ID of the source node is extracted and is considered for insertion if the ID is not present in the neighbor table of receiver. If the ID is not present and the table is full, it evicts a node which has the least send estimate value from the table. If the ID is already in the table of the receiver, the source entry is reinforced with the new routing information.

Table 4.1: Description of a neighbor table entry

Name of the field	**Size (bytes)**	**Description**
id	1	ID of the node
parent	1	ID of the parent node
missed	2	Number of the missed packets from the node
received	2	Number of the received packets from the node
lastSeqNo	2	Sequence number of the latest packet received from the node
flags	1	Status flags (entry new or not, estimations initialized or not, node valid or not)
liveliness	1	Number of send estimations received from the node since the table is last updated
hop	1	Number of hops from the base station
receiveEst	1	Estimated receive quality from the node
sendEst	1	Estimated send quality to the node
nodeType	1	Type of the node in a cluster
clusterheadCost	1	Node's calculated cost of being clusterhead

Implementation Considerations:
- All TinyOS 1 multi-hop routing code is ported to TinyOS 2 following the guidelines in [27].
- A sending queue size of 16 messages is used due to 4096 bytes RAM memory constraint of a MICAz mote.
- The period of broadcasting a routing packet, $T_{routing}$ is set to 10s as stated in Section 4.5.
- The period of running the link estimation algorithms described in this section is determined as 50s.
- ROUTE_INVALID identifier is 1 byte long and has 0xFF value.

4.2 HEED Clustering

The multi-hop routing protocol explained in Section 4.1 provides a best effort service by means of energy-efficiency. It does not offer a prolonged network lifetime which is a crucial requirement for a WSN. A clustering protocol, which manages the network topology with the claim of even distribution of the energy consumption to the nodes, is needed. HEED produces well-distributed cluster heads and compact clusters in the network [9]. In addition, it minimizes the control overheads, since all nodes independently executes and terminates clustering process within a limited number of iterations. Thus, mounting HEED clustering protocol on top of the described multi-hop routing protocol (likewise, TinyOS 1 implementation of HEED was integrated with it and it was called integrated hybrid energy-efficient distributed (iHEED) protocol [28]) is preferred in the implementation. However, the implementation described in this section is not exactly the same. There are some modifications that adjusts HEED to align with the rest of the system. Besides, further enhancements are mentioned at the end of this section.

The implementation is as follows:

- Multi-hop routing protocol and HEED work in coordination for parent selection. To increase performance, clustering and other multi-hop routing information are held in the same table (see Table 4.1). Cluster head cost field in the neighbor table entry is reserved for future improvements. Node type field is required for constructing the tree spanning the cluster head nodes.

- A regular node searches its cluster head in the neighbor table and selects it as a parent after the cluster formation rounds. This parent does not get replaced until the next cluster formation rounds. On the other hand, a cluster head node selects a parent whose node type is a cluster head by following the rules of the multi-hop routing protocol.

- Nodes periodically get into cluster formation rounds. The procedure is initiated by a synchronized timer. Nodes finalize the procedure in a limited number of iterations which varies from node to node. During the cluster formation rounds nodes do not send data packets.

- A regular node is a member of exactly one cluster. It communicates with its cluster head via a single hop. Two initial transmission ranges are defined in the implementation: an intra-cluster transmission range, R_c and an inter-cluster transmission range, R_t. R_c corresponds to an initial intra-cluster power level, P_c and R_t to an initial inter-cluster power level, P_t. Inter-cluster routing must be able to transmit to higher ranges, so $R_t > R_c$ and $P_t > P_c$.

- Inter-cluster routing is realized by the multi-hop routing protocol. Intra-cluster routing is based on building clusters by selecting cluster heads and is realized by forwarding multi-hop traffic to elected cluster heads.

- Two parameters are the actors of the cluster head election process. The primary parameter is used to form an initial set of cluster heads by determining a likelihood of announcing node willingness to become cluster head. The secondary

parameter is used to finalize the election, if two or more nodes in a small area announce their willingness to become cluster head. The primary parameter is determined by the the residual energy of the node. In the implementation, battery voltage, which is measured by the MicaZ analog-to-digital converter (ADC), is used for representing the node's residual energy. The secondary parameter is an estimate of the intra-cluster communication cost which is the number of regular neighbor nodes. In other words, it is the proximal regular node density of the candidate cluster head.

- A probabilistic approach is used to elect the cluster heads. Nodes with higher battery voltage has more chance of being elected compared to regular nodes. The likelihood of announcing itself as a tentative cluster head of a node is:

$$\alpha = \begin{cases} c \times (1-k) & \text{if } V < V_{min} \\ c \times \left(k \times \frac{V - V_{min}}{V_{max} - V_{min}} + (1-k)\right) & \text{if } V_{min} \leq V \leq V_{max} \\ c & \text{otherwise} \end{cases} \tag{4.6}$$

  where V is the current battery voltage of the node, V_{min} is the reference minimum battery voltage, V_{max} is the reference maximum battery voltage, k is a constant that is user to limit the number of clustering iterations and c is a constant that is used for limiting the number initial cluster head announcements.

- ClusteringM component is responsible for realizing the HEED protocol in TinyOS. Clustering packets are broadcast if required, other node's cluster head announcements are managed and cluster heads are selected by ClusteringM component. The clustering packet broadcast during the cluster formation rounds has four fields: ID of the source node, node type, ID of the source node's cluster head and (if its node type is a cluster head) cost of the being cluster head for the source node (see Figure 4.5).

- ClusteringM component interacts with MultiHopLEPSM component through RouteControl and RouteSelect interfaces in TinyOS.

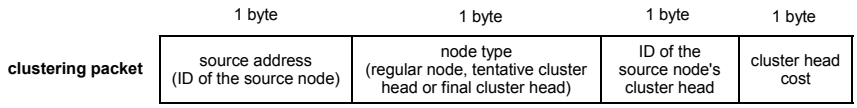

Figure 4.5: Structure of the clustering packet

- Cluster head announcements are broadcast using P_c.

Table 4.2: Description of a final cluster head announcement table entry

Name of the field	**Size (bytes)**	**Description**
addr	1	ID (or address) of the node
cost	1	Communication cost (or number of the regular node connections)

- Each node first announces itself as tentative cluster head. Once it happens, it continues to announce as such during the iterations. At the last iteration if it did not receive any final cluster head announcements before, it announces itself as a final cluster head.

- Each node stores the final cluster head announcements which have the least intra-cluster communication costs. In Table 4.2, details of an entry stored in the final cluster head announcement table can be found.

- The process repeated in each iteration by all nodes during the cluster formation rounds can be seen in Algorithm 1.

Algorithm 1 Clustering algorithm, iteration i

```
if α < 1.0 then
  if no tentative cluster head announcement is received before and the node is not
a tentative cluster head then
    generate a random value
    if randomvalue < α then
      broadcast tentative cluster head announcement
    end if
  else if the node is a tentative cluster head then
    broadcast tentative cluster head announcement
  end if
  α ← α × 2
else
  if the node is a tentative cluster head or the node waited for this iteration then
    if at least one final cluster head announcement is received then
      select the node with the least number of regular node connections as cluster
head among them
      initialize transmission power level for multi-hop data packets to P_c
    else
      broadcast final cluster head announcement
      initialize transmission power level for multi-hop data packets to P_t
      signal the end of cluster formation rounds
    end if
  else
    wait for the next iteration
  end if
end if
```

- HEED is guaranteed to terminate in O(1) iterations [9].

- The probability of two cluster heads lying within R_c of each other is very small [9].

Improvements:
- Synchronization between the nodes getting into cluster formation rounds is not ensured by cluster heads of the previous round announcing not being one anymore as in iHEED. This may decrease the chances of the regular nodes declaring themselves

cluster heads, since the timing errors may put some of the nodes one or more iterations behind in this kind of a synchronization mechanism. Besides, packets that announce being not cluster head anymore may drop and this may introduce serious synchronization problems. Implementation of a more reliable synchronization mechanism is explained in Section 4.4.
- Transmission power adaptation process described in Section 4.3 puts serious effort in order to decrease node's energy consumption by decreasing its P_c or P_t.
- Final cluster head announcement is broadcast once more after the cluster formation rounds, since other nodes may not be able to receive the first one. This further decreases the probability of two cluster heads lying within R_c of each other.

Implementation Considerations:
- Knowing the facts that ADC for the battery voltage may not be useful due to its coarse granularity and the accuracy of the computed ADC result is not always guaranteed as pointed in [28], battery voltage is used for determination of likelihood of being cluster head instead of a credit-point system. This is due to the fact that determination of an effective current consumption, which is actually the heart of the reliability of a credit-point system, is nearly impossible in this complex and non-deterministic implementation of WSN.
- MICAz mote practical operation is in between 2.7V and 3.6V [20]. It is experimentally seen that MICAz motes can operate under these voltages. Hence, V_{min} is selected as 2.7V.
- V_{max} is selected as 3.0V, since it is experimentally seen that two serially connected AA batteries can produce 3.4V at most.
- c is selected as 0.03, k is selected as 0.9994. If a sensor node's battery voltage is less than V_{min}, α is found as 0.000018 using Equation 4.6. Since α is doubled at the end of each iteration and cluster formation round ends when α reaches a value bigger than 1 (see Algorithm 1), the iteration count for a sensor node having low battery voltage is limited to 16. If a sensor node's battery voltage is greater than V_{max}, α is found as 0.03 using Equation 4.6. Likewise, the iteration count for a sensor node having high battery voltage is limited to 6. Lastly, if a sensor node's battery voltage is in between V_{min} and V_{max}, then α is calculated as a value between 0.000018 and 0.03 using Equation 4.6. Thus, its iteration count is limited to a value between 6 and 16.

4.3 Transmission Power Adaptation

The multi-hop routing protocol and HEED do not resolve the interference problem occurs between distant nodes while sending data packets. Besides, using a high transmission power level between close nodes needlessly is energy wasting. A simple transmission power adaptation protocol, which tries to decrease this high power level is introduced in the implementation.

At the end of cluster formation rounds, nodes initialize their transmission power levels as described in Section 4.2. If a node is elected as cluster head, it initially uses P_t. If it remains a regular node, it initially uses P_c. Both node types set their initial power level as the highest possible power level, P_h and they set the minimum power level available for the MicaZ motes as the lowest possible power level, P_l. A child node sends three consecutive transmission power adaptation packets to its parent using transmission

power level, P:

$$P = \lfloor \frac{P_h + P_l}{2} \rfloor. \tag{4.7}$$

If the parent node receives any of these packets, it acknowledges the sender node for each packet individually. While in action, the parent node also use P for the packets to be sent as acknowledgements. If two out of three acknowledgements are received back, the child node replaces its P_h with P. If one or no acknowledgement is received, the child node replaces its P_l with P. And then it recalculates P using Equation 4.7 in order to find the new power level to be used for sending new three transmission power adaptation packets to its parent. The procedure stops and signals the end of transmission power adaptation process when the difference between P_h and P_l is equal to one or in other words P can not be changed anymore. Since P_t and P_c are constant preset transmission power levels, this procedure ends in O(1) iterations.

	1 byte	1 byte	1 byte
transmission power adaptation packet	source address (ID of the source node)	acknowledge indicator (acknowledge or not)	transmission power level

Figure 4.6: Structure of the transmission power adaptation packet

TxPowAdaptationM component is responsible for realizing the power adaptation in TinyOS. It interacts with the MultiHopLEPSM component through RouteControl and ClusteringM component through ClusteringNotify interfaces. In Figure 4.6, fields of the transmission power adaptation packet are shown: ID of the source node, packet acknowledge indicator and transmission power level of the packet.

Implementation Considerations:
- Queue implementation is an important part of the introduced transmission adaptation protocol. A parent node can have many child nodes. Taking into consideration that each child node sends three transmission power adaptation packets, the parent node carries a serious communication burden. An acknowledgement packet needs to be sent for each transmission adaptation packet along with the packets need to be sent for the node's own transmission power adaptation. Thus, a sending queue of transmission adaptation packets with size 10 is implemented.
- Since the regular nodes do not attempt to change their parents till the next cluster formation rounds and the implementation discards node mobility, transmission power adaptation algorithm is applied only once after a cluster formation round. However, this is not feasible for cluster head nodes, since they may change their parents frequently depending on the link conditions. When a cluster head node switches to a new parent, the last transmission power level may not be sufficient. Thus, the cluster head signals parent re-selection so that transmission power adaptation component, TxPowAdaptationM resets its transmission power level to P_t.
- There are 32 transmission power levels supported on a MicaZ mote [20]. P_t is selected as 21st highest transmission power level. P_c is selected as the 16th highest transmission power level. Thus, transmission power adaptation procedure lasts for 5 iterations in a cluster head node and 4 iterations in a regular node.

4.4 Time Synchronization

In distributed systems like WSNs, data from each sensor node may be used to produce a meaningful result at a central station. At this point, mapping of the collected sensor data precisely with the time information turns out to be crucial. Also a time slotted communication may require strict synchronization between nodes. This way, nodes can accurately schedule their send, listen and sleep periods. However, time synchronization in real-world deployments has major difficulties. Nodes suffer from clock drift and skew problems (see Figure 4.7 which illustrates an example for mote A and mote B), because processors on the motes do not run at exactly the same speed. This is due to the fact that oscillators may alter in production phase and their oscillation frequencies are significantly affected by external conditions such as temperature. Thus, the clocks which count these oscillations have slight differences. These differences cumulate over time and clock drift problem occurs. Also some nodes might be powered up some time later than the other nodes in the network and their clocks may be behind with a non-ignorable time offset. In order to reduce these problems to a reasonable level, a precise, accurate and fast time synchronization protocol between network nodes is desirable.

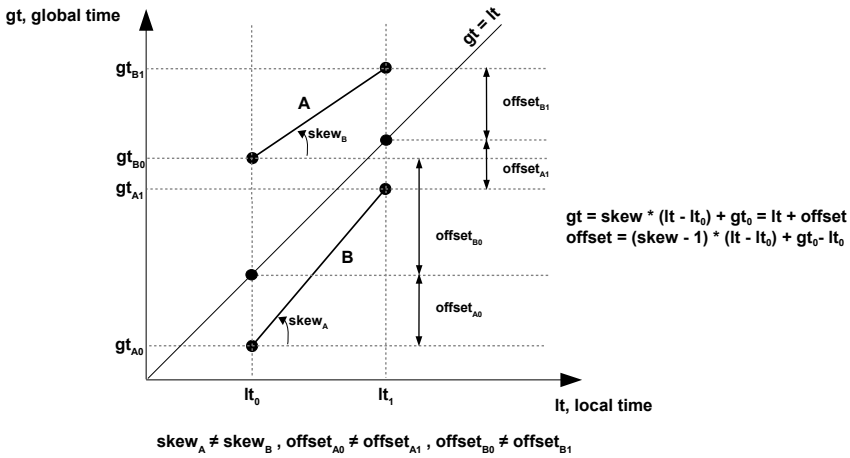

Figure 4.7: Clock drift and skew problems

The TinyOS 2 release includes a library that implements FTSP, which is an ad-hoc, multi-hop time synchronization protocol for WSNs as described in Chapter 2. It timestamps the radio messages in low layers of radio stack and eliminates the radio channel access time required in carrier sense multiple access with collision avoidance (CSMA/CA) MAC protocol. In this section, the implementation of a time synchronization protocol, which is heired from FTSP and best suits the precision/swiftness needs of the whole system, is presented. Also PulseSync protocol, which is described in Chapter 2, is a source of inspiration for the implementation.

In Figure 4.8, the flow of data through the processes, which realize the time synchronization scheme in the implementation, can be seen. Details of these processes are explained below. The base station is selected as the fixed root of the synchronization

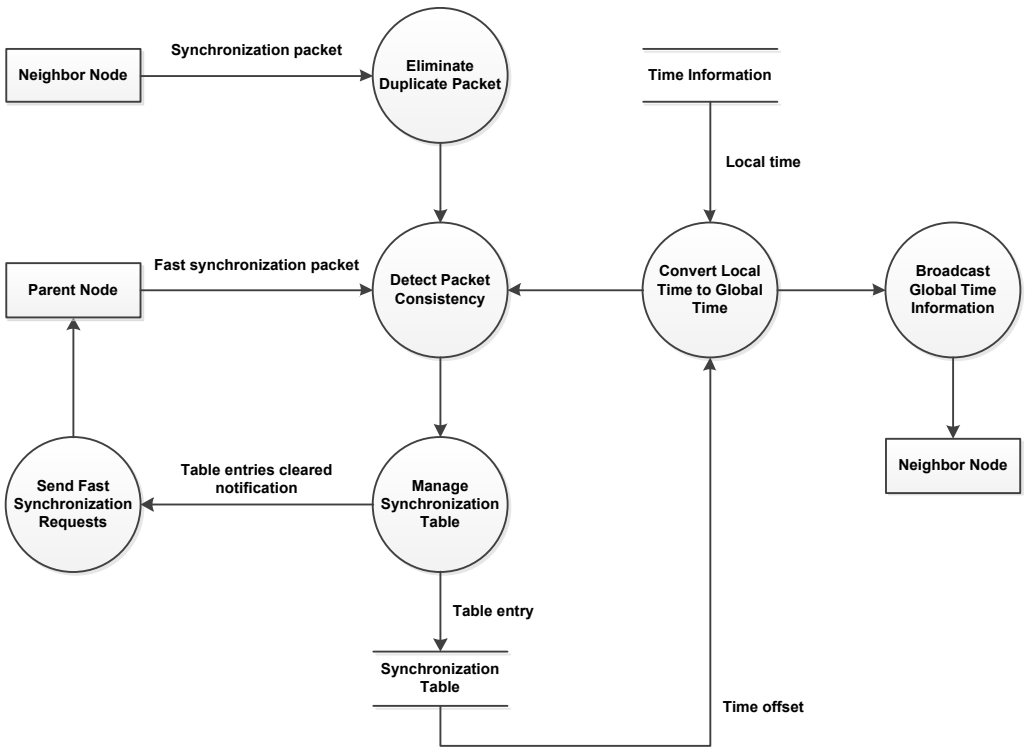

Figure 4.8: Time synchronization protocol data flow diagram

protocol, since it may be powered by an external resource and the base station node can receive the global time information directly from the gateway. A synchronization packet is sent out from the root node with the period of time synchronization packet broadcasts, T_{sync} by TimeSyncP component in TinyOS. The packet has 4 fields: ID of the synchronization root node, ID of the sender node, sequence number of the synchronization packet at the root node and global time of an event (see Figure 4.9). Before a synchronization packet is broadcast by the root node, the global time of an event just before transmission, gt_{evnt_s} is assigned to global time field of the packet. Local time of this event, lt_{evnt_s} is passed to the TinyOS synchronization message sender component and it stores this value to the related field of the TinyOS message header. In order to compensate the transmission and internal processing delay, at the radio stack level TinyOS timestamps this packet exactly when the last byte of this packet is being transmitted into the medium, lt_{snt_s} [29]. At the moment the first byte of this packet is received from the medium by any node, lt_{rcvd_r}, TinyOS timestamps this packet. Using these records in the received packet and ignoring the propagation time, the receiver node knows the time of the event at the sender node in terms of its own local time, lt_{evnt_r}:

$$lt_{evnt_r} = lt_{rcvd_r} - (lt_{snt_s} - lt_{evnt_s}) \tag{4.8}$$

After extracting lt_{evnt_r}, it stores the value as the local time of the event at the receiver node. Then, the ID of the root is checked whether the packet is originated from the root node or not. Also sequence number generated by the root is checked in order to eliminate duplicate packets. If the sequence number of the packet is less than or equal

	2 bytes	2 bytes	1 byte	4 bytes
time synchronization packet	root address (ID of the root node)	sender address (ID of the sender node)	sequence number	global time (time of an event at the root node)

Figure 4.9: Structure of the time synchronization packet

to the last received packet's sequence number, it is marked as duplicate and discarded.

Later on, if the node is synchronized, the consistency of the time information of the non-duplicate and root originated packets is checked. Global time of the event for the node, gt_{evnt_r} is found by applying global time conversion to the stored value of lt_{evnt_r} as in Equation 4.9. If the difference between gt_{evnt_r} and gt_{evnt_s} is above a threshold, the packet is thought as inconsistent. If the inconsistency repeats for three successive packets, synchronization table is cleared since the node knows that it is not synchronized and the values in the table are not valid anymore. If the difference is not above a threshold or the node is not synchronized, the synchronization table and time conversion parameters are updated accordingly with the timing information inside the received packet.

Entry state, local time and time offset values of the synchronization packets are stored in the synchronization table (see Table 4.3). Entry state indicates whether the table slot is empty or not. Local time is stored such that the oldest entry in the table can be found later on. Because the oldest entry is replaced with the newest entry while updating the table. Lastly, time offset, ot_{evnt} which is calculated as:

$$ot_{evnt} = gt_{evnt_s} - lt_{evnt_r} \tag{4.9}$$

is used for time conversion. At any time, if there are more than three entries in the table, the node is assumed to be synchronized. A synchronized node forwards each non-duplicate packet it receives as soon as possible in order to flood the synchronization packets to the network. If a node is going to forward a synchronization packet, it makes use of a new event and fills the packet fields regarding its own synchronized time.

Table 4.3: Description of a time synchronization table entry

Name of the field	**Size (bytes)**	**Description**
entry state	1	State of the table slot (empty or not)
local time	4	Local time of the entry insertion into the table
time offset	4	Offset used for time conversions

To manage time conversion, the values stored in the synchronization table are evaluated. After adding a new entry to the table, the average of the local offset, $ot_{evnt_{avg}}$ is calculated. Local time to global time conversion is done by using this offset value as:

$$gt = lt + ot_{evnt_{avg}} \tag{4.10}$$

where lt is the local time of the node to be converted, gt is the global time according to the node. Also, global time to local time conversion is done as:

$$lt = gt - ot_{evnt_{avg}} \tag{4.11}$$

At the startup or if synchronization is lost for any reason during operation, the node does not need to wait for at least three T_{sync}. It eventually starts the fast synchronization procedure. An unsynchronized node waits for a duration, $T_{syncwait}$ to receive a synchronization packet if it is not synchronized. If any non-duplicate packet is received, it evaluates the packet and waits for another $T_{syncwait}$. This is repeated until at least three valid packets is received and the node gets synchronized. If no message is received after $T_{syncwait}$, the node clears entries and broadcasts synchronization requests with a time synchronization request sending period, $T_{syncrequest}$ (see the structure of a synchronization request packet in Figure 4.10). The reason is explained in Section 4.5.

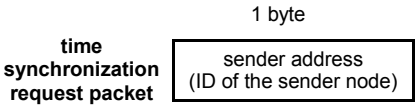

Figure 4.10: Structure of the time synchronization request packet

If a synchronization request is received by any synchronized node, it sends eight consecutive synchronization packets with its own node ID as the root ID. The packets are separated by an interval, $T_{fastsync}$. Evaluating the root ID field, the unsynchronized node can distinguish these packets as fast synchronization packets. Then it gets synchronized after receiving three or more of these packets. If three fast synchronization packets are not received in sequence for any reason, the node restarts the fast synchronization procedure.

Improvements:
- Rates of synchronization packets broadcast by root node at startup are variable. First, a synchronization packet is broadcast every 4 seconds. After four packets are sent with the latest rate, the rate decreases. First, rate decreases to every 8 seconds, then 16 and 32 seconds. After 240 seconds (or sixteen packets are sent), the root starts to broadcast a packet every T_{sync}, which is the stable period for synchronization updates. The aim of the described mechanism is to synchronize the network nodes as fast as possible at system startup.
- Fast synchronization is a new feature embedded in the implementation.

Implementation Considerations:
- In the implementation, T_{sync} is selected as 60s as stated in Section 4.5, $T_{syncwait}$ as 16s and $T_{syncrequest}$ as 96ms.
- FTSP compensates for clock skews of participating nodes via linear regression. This is not implemented because it is experimentally observed that for some reason using skew on the deployment increases the numerical computation errors over time. Instead 32khz clock of the microprocessor is used in order to increase the precision and decrease the effect of numerical errors on synchronization. Actual rate is approximately 28.8khz

[30]. Thus, if a clock is 1 tick behind, it is getting approximately 34.7 microseconds behind. Say a node always gets behind 1 tick per second (which is practically a big error). Getting 60 ticks behind in T_{sync} of 60s means getting approximately 2 ms behind. Even this error is in the acceptable bounds of the implementation which are defined in Section 4.5.
- FTSP is an in-network time synchronization protocol. A regular node in the network is allowed declare itself as the root. In the implementation only the base station is allowed to be the root.
- In FTSP all nodes wait for their periodical timer to fire in order to forward the synchronization packets after they receive them. PulseSync forwards them immediately. In the implementation, neither is applied. First the received packets are evaluated, synchronization table is updated, a new event is generated and lastly they are forwarded.
- $T_{fastsync}$ is selected as 1s.

4.5 Radio Sleep and Event Scheduling

Sleep scheduling is an important part of a WSN implementation because if sensor nodes having limited energy resources are active most of the time, they will deplete their entire energy in a few days, maybe hours. In order to lengthen sensor nodes' lifetime from days to months or years, a low duty cycle is often needed to be defined for some hardware components such as the radio transceiver unit.

Radio transceiver unit is often the most energy consuming hardware component in a sensor node. Energy consumption due to idle listening, especially when the traffic load is light, has a prominent share in the total energy consumption of the transceiver unit. Therefore, sensor nodes' transceiver units must be put into sleep for the periods sensor nodes unnecessarily listen to the channel. However, sleep scheduling scheme must be designed carefully in order not to end up with loss of communication capabilities. Sensor data delivery to the sink node must not be affected while turning off the transceivers of the intermediate nodes. In addition, the network connectivity for other functionalities like routing or clustering must not be broken. In order to stay connected or in other words be able to communicate healthily with other nodes in their neighborhood, every node needs to know when to sleep or wake up. Thus, talking about sleep scheduling is irrelevant if time synchronization is not available in a WSN. This is one of the reasons why time synchronization is implemented in this thesis work as stated in Section 4.4.

MAC layer sleep scheduling techniques are useful for WSN implementations. Especially TDMA scheduling, which offers fixed slots to each node for receiving and transmitting, can remove energy losses due to the idle listening if nodes sleep whenever they are not subject to interaction with another node [31]. However, TDMA layer approaches are not feasible for the implementation in this thesis work. First, they require very accurate time synchronization, which is hard to obtain in a multi hop environment. Second, TinyOS MAC layer implementation of CSMA/CA needs to be replaced for the CC2420 radio hardware, which requires deep knowledge of the actions and responses of the hardware in several situations. Third, buffering becomes a necessity but MICAz motes has a very limited memory constraint which is 4096 bytes. Fourth, how nodes

agree on fixed time slots comes out as a problem to be solved. Lastly, accuracy of the time synchronization is vital but it may vary too much since depends on too many factors on real motes like hop count, temperature, propagation time and computation errors. Considering these facts, a sleep scheduling scheme is devised at the application layer in this thesis work.

Features specific to this implementation are taken into account while designing a custom sleep scheduling scheme. There are five packet types in the implementation: routing (see Section 4.1), clustering (see Section 4.2), transmission power adaptation (see Section 4.3), time synchronization (see Section 4.4) and data (see Section 4.6). They have different period requirements:

- Routing packets are sent out with a period of routing packet broadcasts, $T_{routing}$.
- Cluster formation rounds are triggered with a long period $T_{clustering}$ and clustering packets are sent with a short period $T_{clusteringit}$ within these rounds. A minimum number of iterations, $N_{clusteringitmin}$ is guaranteed for each node, if the node's battery voltage is V_{max}. Also there exists a maximum number of iterations, $N_{clusteringitmax}$, since iterations are guaranteed to terminate thanks to k constant and V_{min} (see Section 4.2).
- Transmission power adaptation packets are sent with a short period $T_{txadaptit}$ after each cluster formation round. Transmission power adaptation procedure is guaranteed to terminate in a maximum number of $N_{txadaptitmax}$ iterations. Also the minimum number of iterations to terminate is $N_{txadaptitmin}$.
- Time synchronization packets are sent with varying periods. The period remains fixed at T_{sync} after the transient startup period. Note that, T_{sync} is the maximum period, with which time synchronization packets are sent.
- Data packets are sent with a period T_{data}.

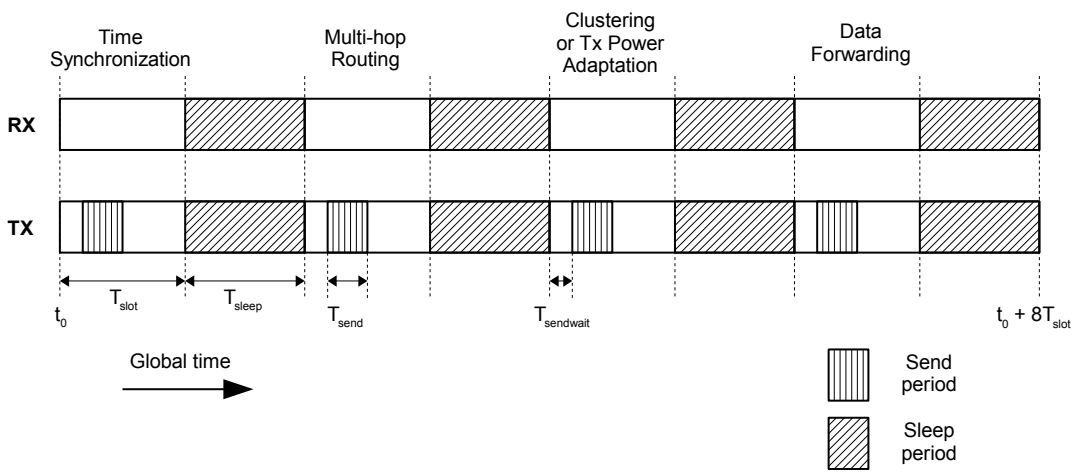

Figure 4.11: A frame in time

The periods mentioned above must be a multiple of T_{frame} for proper operation. Taking this into consideration, a frame with duration T_{frame}, which is smaller than all these periods, is defined in time at the application layer where all types of packets can be sent or received without causing disturbance to each other.

A frame is divided into eight slots (see Figure 4.11). Four of them are active periods, four of them are sleep periods. Packet transmissions are done in active periods. One time slot is reserved for each of the packet types except that one of them is reserved for both clustering and transmission power adaptation packets. Each slot has a duration T_{slot}. After the slot, in which radio units of all nodes are active, there exists a sleep interval (a synchronized node is never allowed to transmit or receive packets in this interval) with duration T_{sleep}. Within a frame, radio units of nodes are not always active in the slots defined. They are only activated with the mentioned periods, which are bound to the related packet types. That is to say, when the time for the related packet comes, the radio units of the nodes are activated in order to allow the nodes to communicate. Thus, keeping in mind that radio units of the nodes may not be active in active slots of each frame, during the operation of the sensor nodes several combinations of these active time slots will naturally be observed in a frame. As a result, assuming that the sensor node remains synchronized and does not change its parent for a long time, the maximum and minimum average rate of periods, γ_{avgmax} and γ_{avgmin}, that may occur during this long stable operation can be calculated as:

$$\gamma_{avgmax} = \frac{1}{T_{sync}} + \frac{1}{T_{routing}} + \frac{1}{T_{clustering}} \times \left(\frac{N_{clusteringitmax}}{T_{clusteringit}} + \frac{N_{txadaptitmax}}{T_{txadaptit}} \right) + \frac{1}{T_{data}} \tag{4.12}$$

$$\gamma_{avgmin} = \frac{1}{T_{sync}} + \frac{1}{T_{routing}} + \frac{1}{T_{clustering}} \times \left(\frac{N_{clusteringitmin}}{T_{clusteringit}} + \frac{N_{txadaptitmin}}{T_{txadaptit}} \right) + \frac{1}{T_{data}} \tag{4.13}$$

Using Equation 4.13 and Equation 4.12, maximum and minimum average duty cycles, D_{avgmax} and D_{avgmin} can be computed as:

$$D_{avgmax} = \gamma_{avgmax} \times \frac{T_{slot}}{T_{frame}} \times 100\% \tag{4.14}$$

$$D_{avgmin} = \gamma_{avgmin} \times \frac{T_{slot}}{T_{frame}} \times 100\% \tag{4.15}$$

In Figure 4.11, it can be seen that the first slot is reserved for time synchronization packets. The order of the selection of the slots is not arbitrary. By receiving a synchronization packet is during the first slot, a node may become synchronized and later on it can start the sleep scheduling procedure, communicate in the related slot properly and timestamp the data packets accurately. By receiving a routing packet is during the second slot, a node can discover its neighbors or select a parent, which directly affects clustering and data forwarding procedures. The last slot is for data packets because all other procedures are preconditions for the delivery of data packets to the central station properly in an energy-efficient way.

Idle listening problem is overcome with this kind of an approach. Thanks to synchronization module, all nodes enter the slots approximately at the same time and communicate in related operation's reserved slot. However, drawbacks of this approach must not be ignored. Active-sleep mode transition delays of the radio unit,

errors in time and latencies introduced while hopping may still cause packets to be dropped frequently. In order to tolerate them, nodes start sending their packets after waiting for a time interval, $T_{sendwait}$ at the beginning of each slot. Then, packets are sent uniformly at random moments in a T_{send} interval. This approach at the application layer aims to minimize the encountering rates of back-off periods in TinyOS CC2420 Radio CSMA/CA implementation [32]. The rest of the slot is only reserved for listening in order to tolerate network latencies.

Designing sleep scheduling as described above also has the benefit of relaxing the concurrent execution of all related functions and tasks in the implementation, since the load of their execution is distributed in a transmission time interval, which is defined as a frame in this implementation.

In TinyOS an event scheduling module is realized in order to implement radio sleep or other events scheduling on the system. All nodes independently schedule its internal events. So, the harmony in each node is ensured by this module. At the heart of the module, an asynchronous timer, a slot timer and a send timer reside (see Algorithm 2).

Algorithm 2 Event scheduling algorithm - asynchronous timer

```
if asynchronous timer fires then
  if current event is a communication event then
    turn the radio on
  end if
  if node is synchronous then
    start slot timer
  end if
  if current event is a sending event then
    start random send timer
  else if current event is not idle then
    signal current event
  end if
  get global time from time synchronization module
  find the next event
  getting the position of the current event as reference, calculate the global time of
the next event
  assign next event to current event
  set local asynchronous timer to fire at the global time of the event
end if
```

Algorithm 2 Event scheduling algorithm - slot timer

```
if slot timer fires then
  turn the radio off
end if
```

Algorithm 2 Event scheduling algorithm - send timer

```
if send timer fires then
  signal sending event
end if
```

Implementation Considerations:
- $T_{routing}$ is set to 10s.
- $T_{clustering}$ is set to 900s and $T_{clusteringit}$ is set to 1s. $N_{clusteringitmin}$ is computed as 6 if c in Equation 4.6 is set to 0.03. $N_{clusteringitmax}$ is computed as 16 if c and k in Equation 4.6 are set to 0.03 and 0.9994 respectively as stated in Section 4.2.
- $T_{txadaptit}$ is set to 1s. $N_{txadaptitmax}$ is 5 if P_t mentioned in Section 4.3 is set to 21. $\mathrm{N_{txadaptitmin}}$ is 4 if P_c mentioned in Section 4.3 is set to 16.
- $N_{syncstartup}$ is selected to be 16. $T_{syncmin}$ is set to 4s. T_{sync} is set to 60s.
- T_{data} is set to 5s.
- T_{frame}, which is also the minimum period for sending packets, is set to 1s. T_{slot} is 1/8 of T_{frame}. T_{sleep} is also 1/8 of T_{frame}. $T_{sendwait}$ is selected as 16ms. T_{send} is selected as 48ms.
- $T_{syncrequest}$ is selected as 96ms. This way, the necessity of catching an active period of the parent node as soon as possible is satisfied.
- D_{avgmax} and D_{avgmin} in Equation 4.12 and in Equation 4.13 are computed as 4.25% and 4.09% respectively using the values stated in this section.

4.6 Accelerometer Data Sampling and Forwarding

The lifetime of a WSN, which monitor the environment, decidedly depends on the sensor data sampling and forwarding rates of the nodes. For this reason, unnecessary sensor data sampling or forwarding must be avoided. Two approaches are realized in the implementation. One of these approaches disregards energy savings from sensor data sampling and forwarding, since it is assumed that central station needs all sensor data samples of each single node. Sensor nodes periodically sample the environmental condition and forward the data packets in the reserved time slots. This way, network's sensor coverage area is continually monitored. The other approach has no intelligent mechanisms which utilize the density of the sensors in an area in order to decrease the sampling and forwarding rates accordingly. In this approach, a cluster head node's radio unit is always active in the reserved slots for data packets, since it is responsible for forwarding data of its own cluster and clusters that resides in the depths of the routing tree. On the other part, a regular sensor node is triggered by a variation in the ambient environmental condition. If there is no activity inside the coverage area of a regular sensor node, its radio unit goes on sleeping in the reserved slot for data packets. This way, a tremendous amount of overall network energy is saved.

In Figure 4.12 the structure of a data packet is shown. There are nine fields in the packet. Timestamp field is reserved for the global time when the sensor data in the packet is sampled. Reading field is reserved for the sensor data. Parent address field is the ID of the original sender node's parent. Battery voltage is the ADC battery voltage reading of the sensor node. Data cost is the total energy cost for the sender node's own sensor data transmissions. Listening cost is the total energy cost for idle listening periods of the sender node. Sleep cost is the total energy cost for sleeping periods of the sender node. Overhead cost is the total energy cost for network maintenance and data forwarding activities of the sensor node. Transmission power level is the latest adapted transmission power level of the sensor node. Cluster head indicator is for tracing whether the node is cluster head or regular node when the packet is sent.

	4 bytes	2 bytes	2 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	1 byte	1 byte
data packet	timestamp	reading	parent address	battery voltage	data cost	listening cost	overhead cost	sleep cost	tx power level	cluster head indicator

Figure 4.12: Structure of the data packet

Implementation Considerations:
- Both two mechanisms fit with the other modules in order that they can be implemented in TinyOS.
- Accelerometer sensors are used as data source in the implementation. The triggering mechanism is implemented as in the movement detection mechanism of the anti-theft application in TinyOS 2 [16].

4.7 Energy Consumption Calculations

In the TinyOS implementation, EnergyCalculatorM module is written for communication related energy dissipation estimations. This module is in interaction with all modules during the operation of the sensor node. If a packet transmission occurs or radio unit changes its state, this module is notified in order that it can estimate the total communication related energy consumption. There are four addends which build up the total communication related energy consumption for a sensor node. These are the consumptions for: transmitting data packets, transmitting overhead packets, listening to the channel and sleeping periods. First one, energy consumption for listening to the channel, where the radio unit is in its active mode, is the most significant among them. In order to achieve its fidelity, TinyOS radio on/off signals are traced to determine the durations of the active and sleep periods. Second, the energy consumption during the transmission of data packets is often the secondary noticeable addend. Its magnitude greatly depends on the application type. If there is a periodic data sampling and forwarding requirement in the application, its cost will be relatively high compared to other applications having no such requirement. Also increasing the rates for data sampling and forwarding greatly increases the number of packets flowing through the WSN, which results in higher consumption for both sending and listening for data packets. Third, energy consumption during transmission of overhead packets, is defined as the summation of wasted energy for forwarding other sensor node's data packets and sending routing, clustering, time synchronization and transmission power adaptation packets. And lastly, energy consumption for sleeping periods is the dissipation of the radio unit during sleeping periods. It is a negligible amount even with an ultra-low duty cycle, since the current draw of the radio unit in sleep mode is insignificant compared to its active modes (see Table 4.4). The calculations are not exact, since the transient periods of the radio hardware for activating sleep, transmission mode or receive modes are totally ignored. But they are valuable for giving strong idea about real-time communication related energy consumptions. The results obtained can be densely used for further improvements.

The overhead and data transmission energy costs, E_{tx} are calculated as:

$$E_{tx} = V \times I_{txlvl_n} \times t_{txduration} \tag{4.16}$$

where V denote the latest sample of the battery voltage, I_{txlvl_n} denote the current drawn during transmission of packets with power level n, $t_{txduration}$ denote the duration of the activity.

$t_{txduration}$ is calculated as:

$$t_{txduration} = \frac{n}{\sigma_{txbitrate}} \tag{4.17}$$

where $\sigma_{txbitrate}$ denote the transmission bit rate of the radio unit, n denote the length of the packet transmitted in bits.

The listening energy cost, $E_{listening}$ is calculated as:

$$E_{listening} = V \times I_{listen} \times t_{listenduration} \tag{4.18}$$

where V denote the latest sample of the battery voltage, I_{listen} denote the current drawn during listening periods and $t_{listenduration}$ denote the listening duration.

$t_{listenduration}$ is extracted using:

$$t_{listenduration} = t_{elapsed} - t_{sleepduration} - t_{totaltxduration} \tag{4.19}$$

where $t_{elapsed}$ denote the elapsed time which represent the interval that the calculation is made for, $t_{sleepduration}$ denote the sleep duration during the subjected interval, $t_{totaltxduration}$ denote the total duration of the packet transmission during the subjected interval.

The sleeping energy cost is calculated similarly:

$$E_{sleeping} = V \times I_{sleep} \times t_{sleepduration} \tag{4.20}$$

where V denote the latest sample of the battery voltage, I_{sleep} denote the current drawn during sleeping periods and $t_{sleepduration}$ denote the sleeping duration.

Implementation Considerations:
- The energy consumption calculations are summed up and total energy consumption values till from the moment the sensor node is powered up are reported to the base station.
- CC2420 transmission bit rate is 250kbps [21]. Thus, $\sigma_{txbitrate}$ is set to 250kbps.
- A look up table is used for the current consumptions, where the values are linearly interpolated using the values in [21] (see Table 4.4).

Table 4.4: CC2420 radio chip's current consumption values

Description	**Current drawn (mA)**
Transmission power level 31	17.4
Transmission power level 27	16.5
Transmission power level 23	15.2
Transmission power level 19	13.9
Transmission power level 15	12.5
Transmission power level 11	11.2
Transmission power level 7	9.9
Transmission power level 3	8.5
Listen (RX mode)	18.8
Sleep	0.02

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 Deployment

In order to validate the final WSN implementation, four experiments were conducted on a testbed in which 8 MICAz motes were used. One of the motes was selected as the base station and it was attached to a PC through USB connection to collect and record data packets sent by the motes in the network. On the PC side, a graphical user interface (GUI) application was developed using Java and JUNG library so that received data packets were processed and real-time network tree was displayed as seen in Figure 5.1 [33][34]. This GUI also had been generating a log file which included the information in data packets sent by the motes during the network operation as seen in Figure 5.2.

The experimental study in this chapter examines all features enabled, sleep scheduling and transmission power adaptation disabled and only transmission power adaptation disabled modes of the implementation as well as the operation of a single mote via its current consumption. Traces of measured battery voltages of the motes; traces of calculated communication related energy consumptions for data packet transmissions, overhead packet transmissions, receive periods and sleeping periods; number of packets delivered to the base station and number of packets lost; and also cluster head election counts of the motes are the metrics used to evaluate the WSN implemented in this thesis work. Oscilloscope screenshots, which are captured while measuring the current consumption of a single mote, are the outputs used to demonstrate and validate the activity of a single mote during the network operation.

5.2 Experiments

Experiment I: 8 MICAz motes were programmed with the whole features explained in Chapter 4 through a MIB520CB programming board. They were placed in arbitrary positions as shown in Figure 5.3. First, base station mote was powered on so that the other motes could receive the time synchronization packets and get synchronized. Then, the remaining motes were powered on. Data packets sent by the motes had been collected for 10 hours. The results were stored in the log file generated by the network monitoring application.

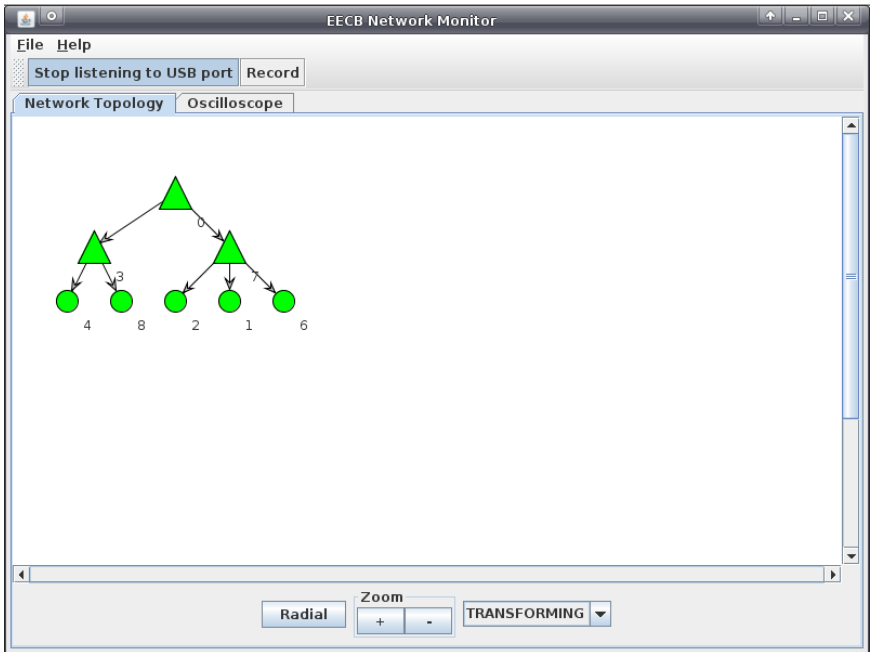

Figure 5.1: Network monitoring application

SrcAddr	OrgAddr	SeqNo	HopCount	Time	Reading	Parent	Battery	DataCost	ReceiveCost	OverheadCost	SleepCost	TxPowLvl	ClusterHead
0	0	2	0	310034	2	0	3.244	0.000037	0.351000	0.000038	0.000000	21	1
0	0	3	0	454049	3	0	3.244	0.000073	0.656000	0.000076	0.000000	21	1
0	0	5	0	598036	4	0	3.252	0.000110	0.960000	0.000114	0.000000	21	1
0	2	6	0	599076	2	0	3.092	0.000035	0.337000	0.000036	0.000000	16	0
0	7	7	0	599273	1	0	2.587	0.000000	0.000000	0.000000	0.000000	16	0
0	0	8	0	742051	5	0	3.252	0.000147	1.266000	0.000152	0.000000	21	1
0	2	9	0	742921	3	0	3.107	0.000070	0.606000	0.000113	0.000000	16	0
0	7	10	0	743259	2	0	2.582	0.000029	0.284000	0.000064	0.000000	16	0
0	0	12	0	886038	6	0	3.252	0.000184	1.571000	0.000263	0.000000	21	1
0	6	13	0	886515	3	0	2.872	0.000064	0.691000	0.000098	0.000000	16	0
0	1	14	0	886814	3	0	2.939	0.000067	0.451000	0.000034	0.000000	16	0
0	7	15	0	887303	3	0	2.614	0.000059	0.527000	0.000094	0.000000	16	0
0	2	16	0	887184	4	0	3.107	0.000105	0.620000	0.000149	0.000000	16	0
0	3	17	0	887343	1	0	3.092	0.000000	0.000000	0.000000	0.000000	16	0
0	0	18	0	1030053	7	0	3.252	0.000221	1.877000	0.000337	0.000000	21	1
0	3	19	0	1030653	2	0	3.092	0.000035	0.494000	0.000082	0.000000	16	0
0	7	20	0	1030898	4	0	2.614	0.000088	0.539000	0.000133	0.000000	16	0
0	6	21	0	1030812	4	0	2.872	0.000097	0.703000	0.000141	0.000000	16	0
0	2	22	0	1031059	5	0	3.107	0.000140	0.635000	0.000195	0.000001	16	0
0	1	23	0	1031361	4	0	2.939	0.000100	0.727000	0.000081	0.000000	16	0
0	0	25	0	1174040	8	0	3.252	0.000257	2.182000	0.000559	0.000000	21	1
0	3	26	0	1174415	3	0	3.115	0.000070	0.783000	0.000118	0.000000	16	0
0	7	27	0	1174852	5	0	2.614	0.000118	0.550000	0.000164	0.000000	16	0
0	2	28	0	1174879	6	0	3.107	0.000175	0.649000	0.000232	0.000001	16	0
0	6	29	0	1174996	5	0	2.872	0.000129	0.717000	0.000174	0.000001	16	0
0	8	30	0	1175033	3	0	2.728	0.000062	0.564000	0.000032	0.000000	16	0
0	0	31	0	1318055	9	0	3.252	0.000294	2.487000	0.000781	0.000000	21	1
0	2	32	0	1318129	7	0	3.107	0.000211	0.670000	0.000317	0.000001	16	0
0	7	33	0	1318221	6	0	2.614	0.000147	0.568000	0.000236	0.000001	16	0
0	8	34	0	1318340	4	0	2.752	0.000093	0.801000	0.000109	0.000000	16	0
0	3	35	0	1318454	4	0	3.115	0.000105	0.804000	0.000207	0.000000	16	0
0	6	36	0	1319344	6	0	2.872	0.000162	0.737000	0.000253	0.000001	16	0
0	0	38	0	1462042	10	0	3.244	0.000331	2.792000	0.000965	0.000000	21	1
0	1	39	0	1462189	7	0	2.953	0.000200	1.029000	0.000199	0.000001	16	0
0	7	40	0	1462543	7	0	2.619	0.000177	0.573000	0.000236	0.000001	16	0

Figure 5.2: Log file generated by the monitoring application

Results

The battery voltages of each mote during the network operation were obtained as in Figure 5.4. The energy consumptions in listening periods for packet receptions were obtained as in Figure 5.5. The energy consumptions for each mote's own data packet transmissions were obtained as in Figure 5.6. The total overhead energy consumptions of each mote for packet transmissions were obtained as in Figure 5.7. The energy consumptions in sleeping periods were obtained as in Figure 5.8. The motes' lost data packet counts during the multi-hop operation and the motes' delivered data packet

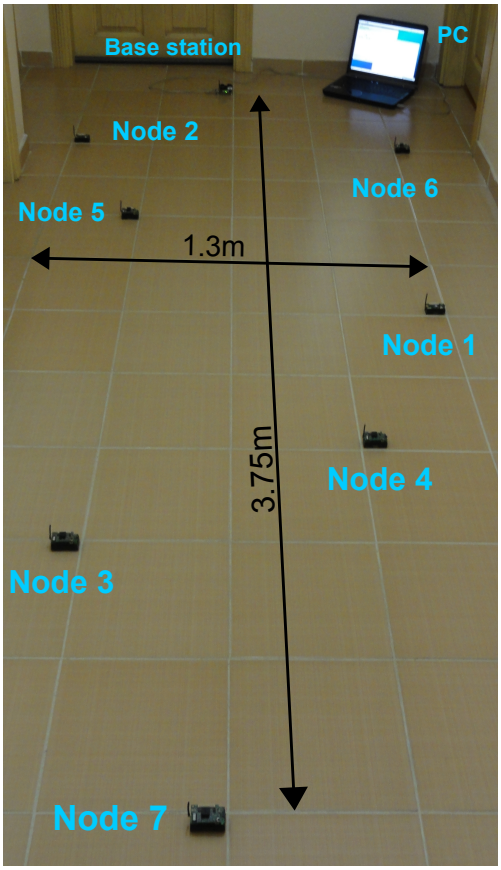

Figure 5.3: Experimental setup

counts to the base station were observed as in Figure 5.9. Cluster head election counts after 40 cluster formation rounds were observed as in Figure 5.10.

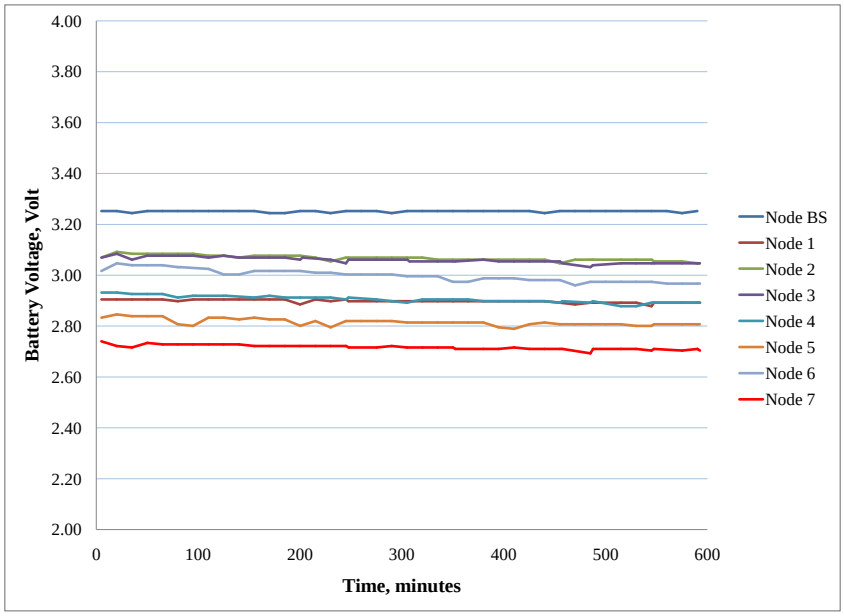

Figure 5.4: Experiment I - Battery voltages

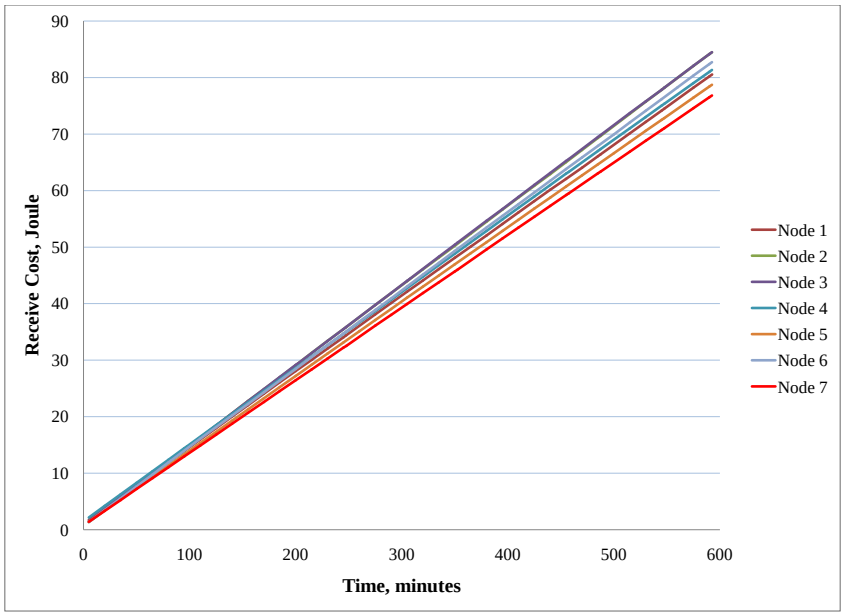

Figure 5.5: Experiment I - Energy consumption in listening periods for packet reception

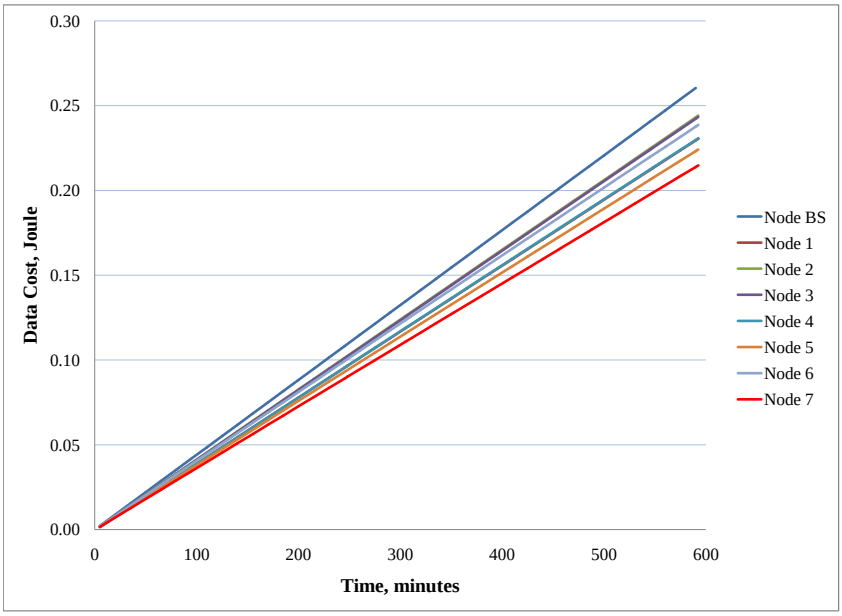

Figure 5.6: Experiment I - Energy consumption for data packet transmissions

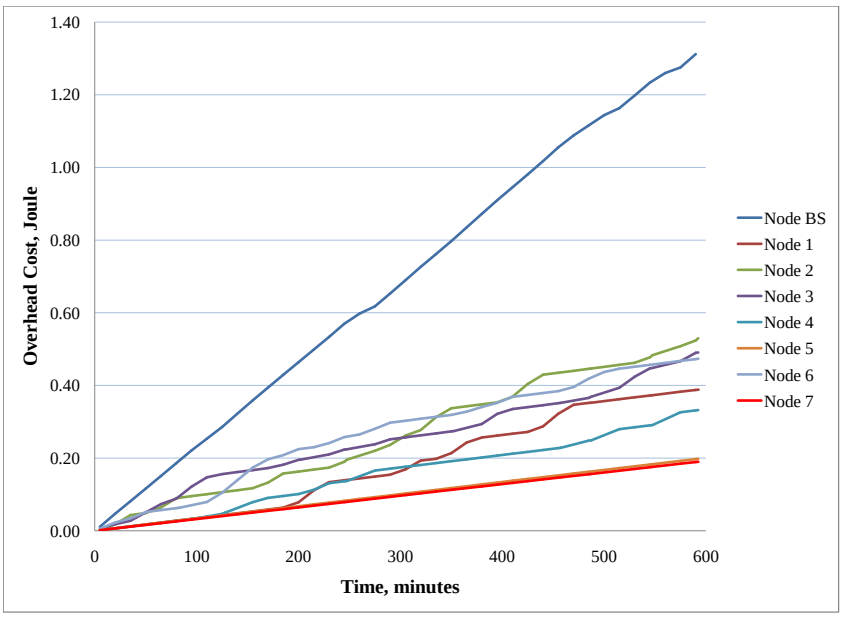

Figure 5.7: Experiment I - Overhead energy consumption for packet transmissions

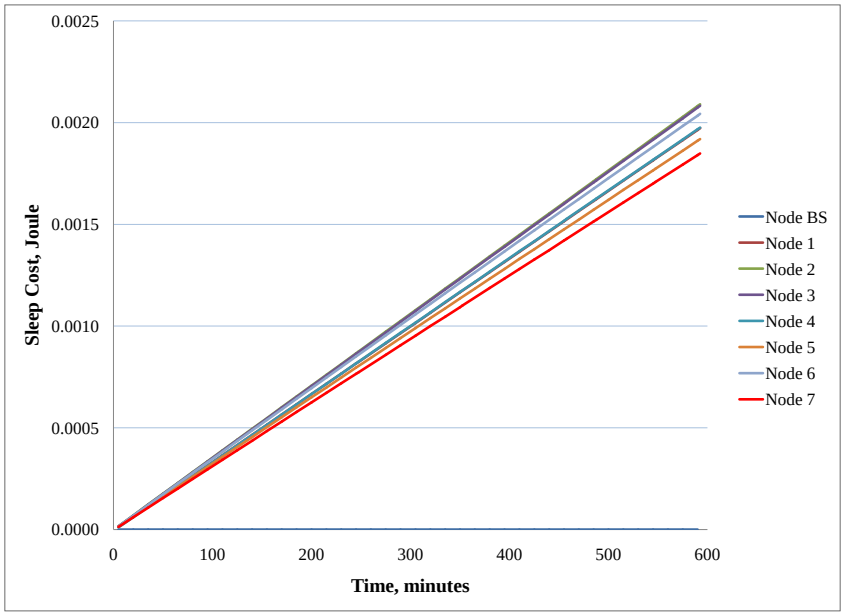

Figure 5.8: Experiment I - Energy consumption in sleeping periods

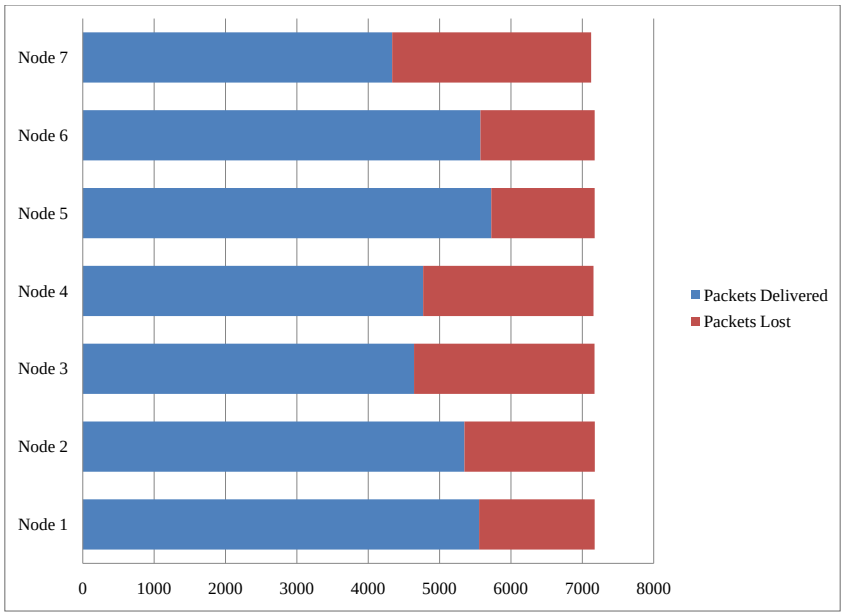

Figure 5.9: Experiment I - Number of the packets lost and delivered to the base station

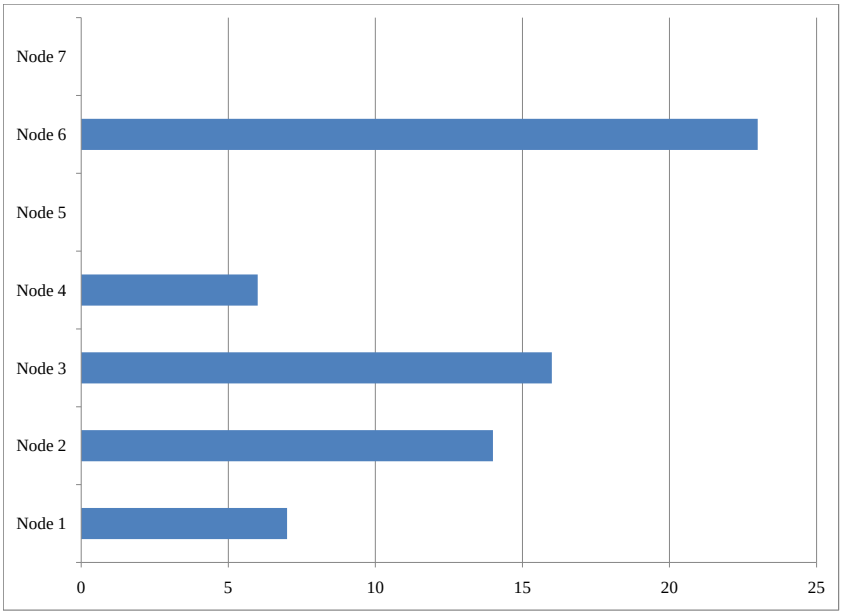

Figure 5.10: Experiment I - Cluster head election counts

Experiment II:

8 MICAz motes were programmed with the sleep scheduling and transmission power adaptation features explained in Chapter 4 disabled. They were placed in the same positions shown in Figure 5.3. Then, the same procedure was applied as in Experiment I.

Results

The battery voltages of each mote during the network operation were obtained as in Figure 5.11. The energy consumptions in listening periods for packet receptions were obtained as in Figure 5.12. The lost data packet counts during the multi-hop operation and delivered data packet counts to the base station were observed as in Figure 5.13.

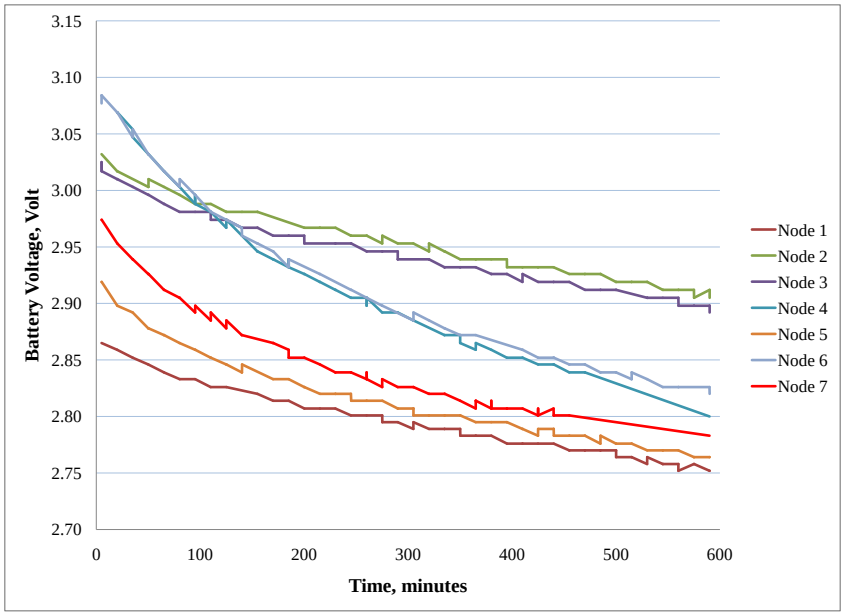

Figure 5.11: Experiment II - Battery voltages

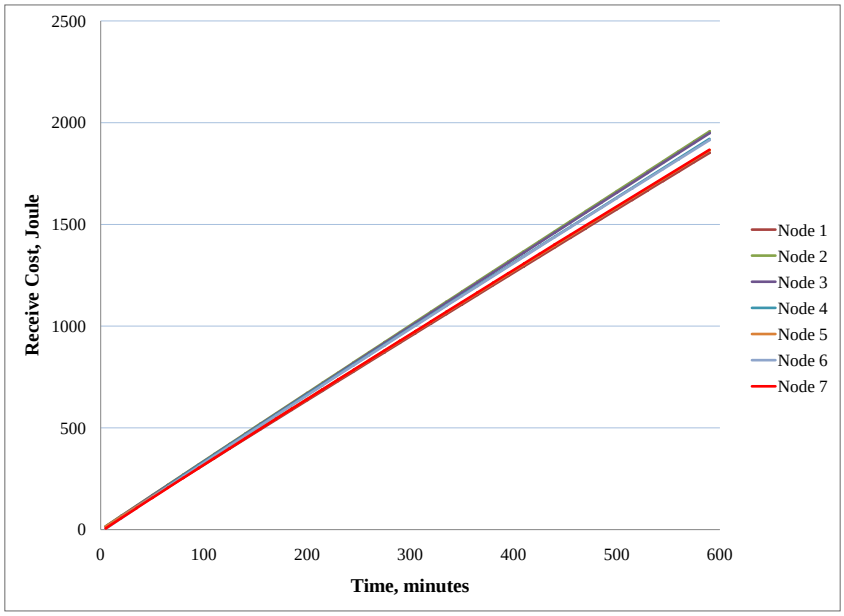

Figure 5.12: Experiment II - Energy consumption in listening periods for packet reception

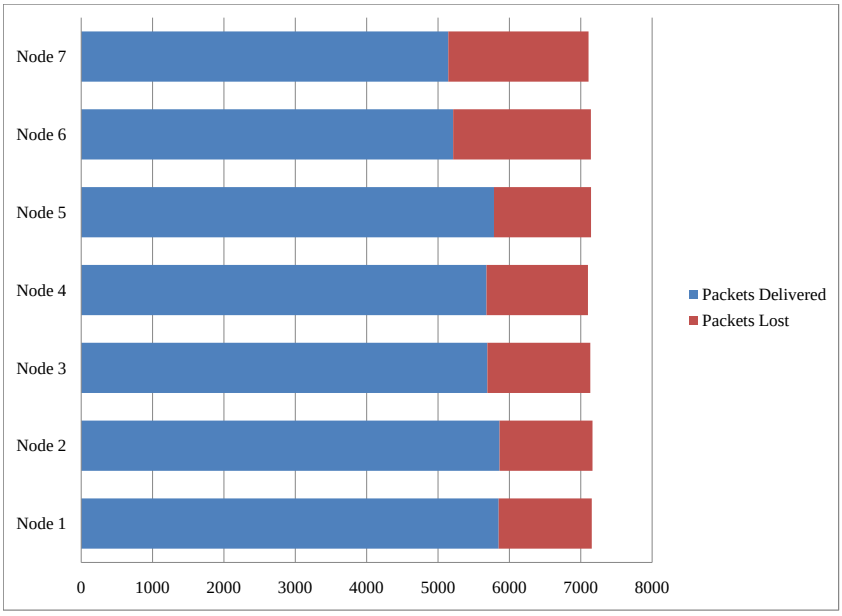

Figure 5.13: Experiment II - Number of the packets lost and delivered to the base station

Experiment III:

8 MICAz motes were programmed with the transmission power adaptation feature explained in Chapter 4 disabled. They were placed in the same positions shown in Figure 5.3. Then, the same procedure was applied as in Experiment I.

Results

The battery voltages of each mote during the network operation were obtained as in Figure 5.14. The energy consumptions in listening periods for packet receptions were obtained as in Figure 5.15. The energy consumptions for each mote's own data packet transmissions were obtained as in Figure 5.16. The total overhead energy consumptions of each mote for packet transmissions were obtained as in Figure 5.17. The motes' lost data packet counts during the multi-hop operation and the motes' delivered data packet counts to the base station were observed as in Figure 5.18. Also transmission power levels used during the network operation in Experiment I and III were observed as in Figure 5.19.

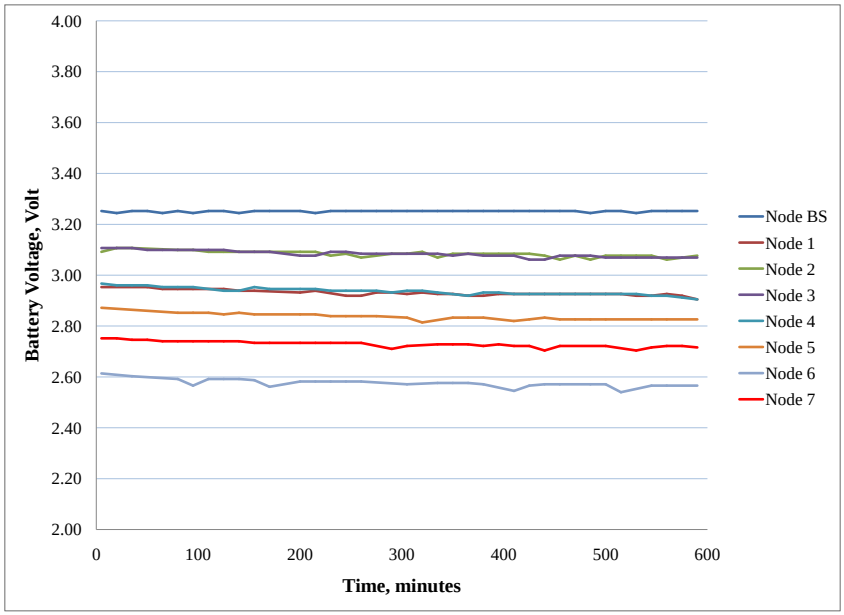

Figure 5.14: Experiment III - Battery voltages

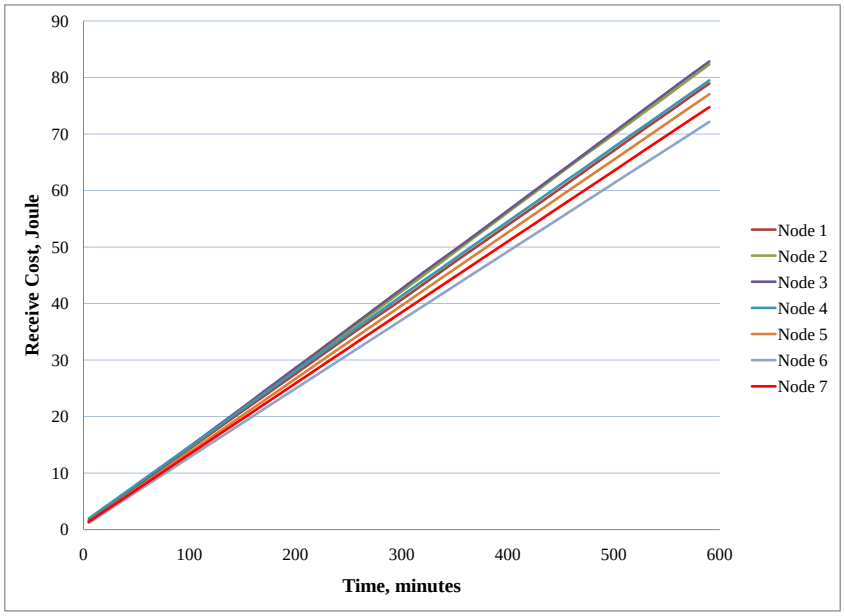

Figure 5.15: Experiment III - Energy consumption in listening periods for packet reception

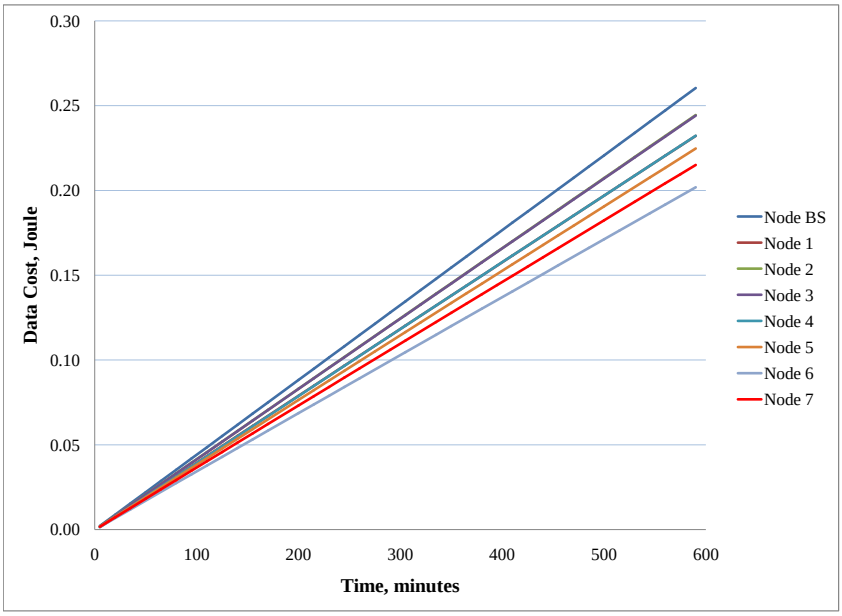

Figure 5.16: Experiment III - Energy consumption for data packet transmissions

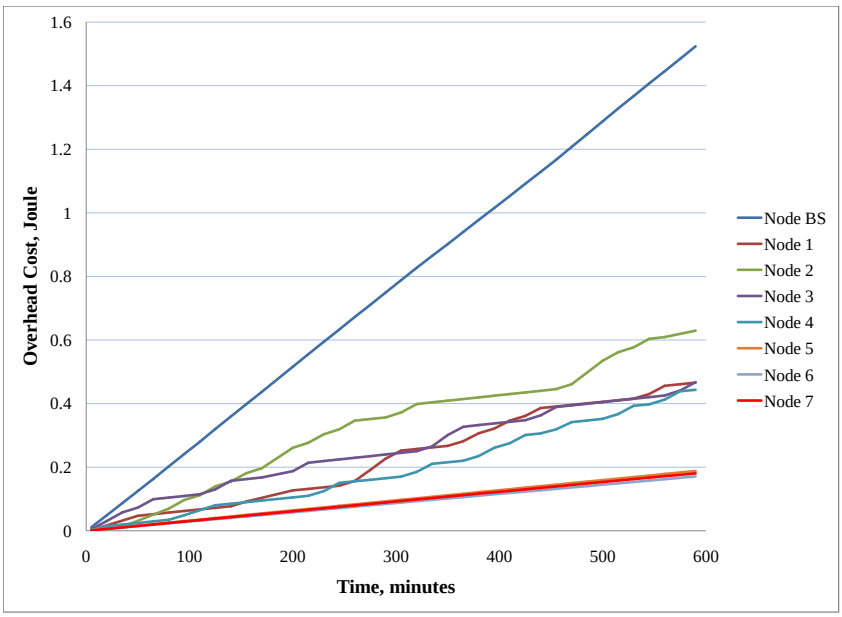

Figure 5.17: Experiment III - Overhead energy consumption for packet transmissions

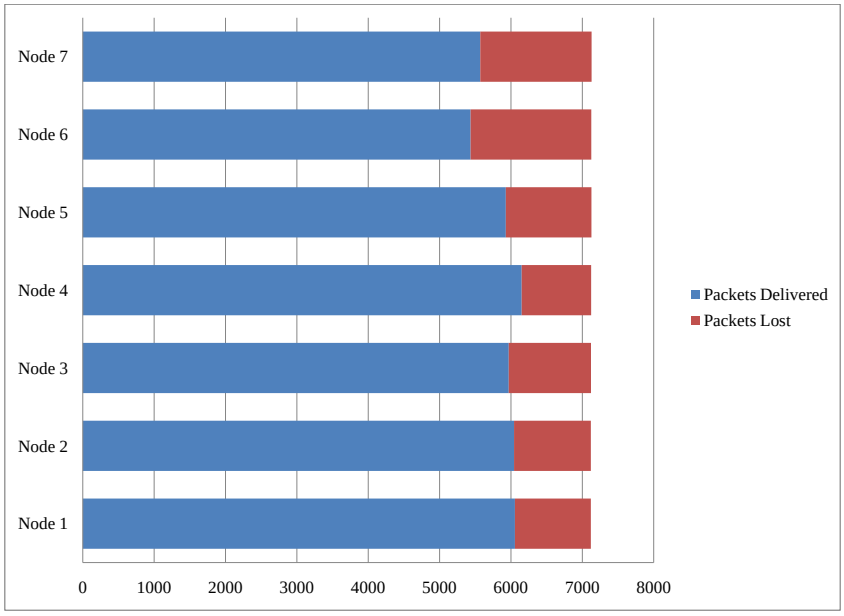

Figure 5.18: Experiment III - Number of the packets lost and delivered to the base station

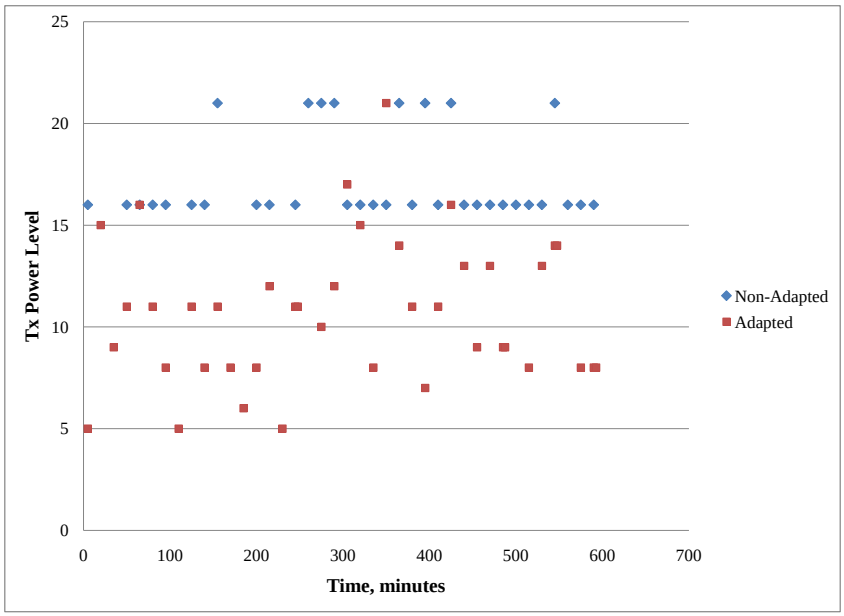

Figure 5.19: Experiment I and III - Adapted and Non-adapted Tx Power Levels

Experiment IV:

8 motes were programmed with the whole features of the implementation. One of the MICAz motes was powered by an external energy supply. The mote was connected in serial with an 1Ω resistor. The voltage difference on the terminals of the resistor were measured continually using an oscilloscope. This way, the current drawn by the mote and duty cycles of the mote resulting from the sleep scheduling were monitored instantaneously.

Results

- In Figure 5.20 4 active and 4 sleeping periods, which are explained in Section 4.5, in a complete frame can be seen clearly. Also it is seen that active period duration between the cursors is measured as 127ms.

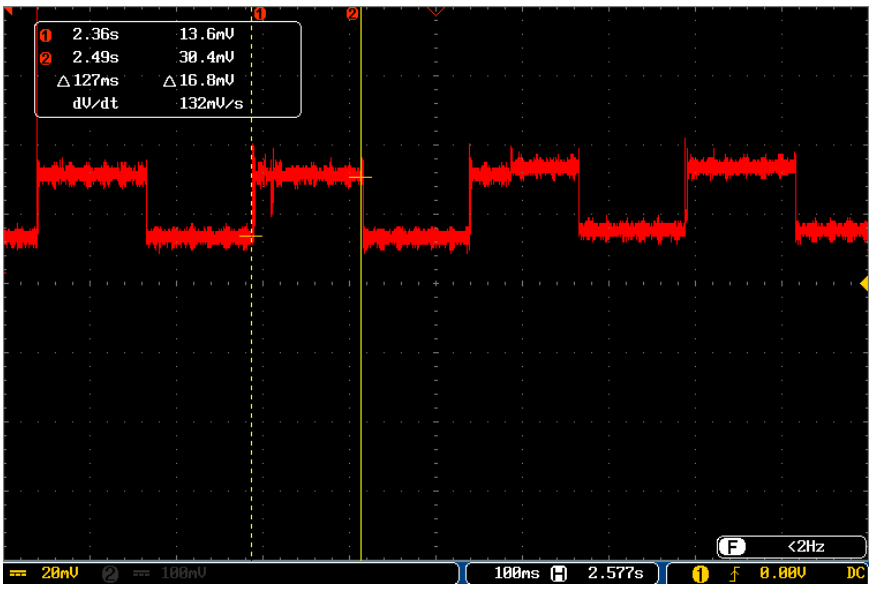

Figure 5.20: Complete frame - Slots

- In Figure 5.21 sleep period duration, which is claimed in Section 4.5, between the cursors is measured as 126ms.
- In Figure 5.22 three consecutive frames at the beginning of the cluster formation rounds can be seen. Different active and sleep period combinations which are claimed in Section 4.5 are observed.
- In Figure 5.23 synchronized operation of the network for 10 seconds is observed.
- In Figure 5.24 the node loses its synchronization and radio sleeping feature of the node is disabled. Thus, all slots become active as stated in Section 4.5.
- In Figure 5.25 after losing its synchronization, it is observed that node starts to send fast synchronization request packets every 96ms as stated in Section 4.5.

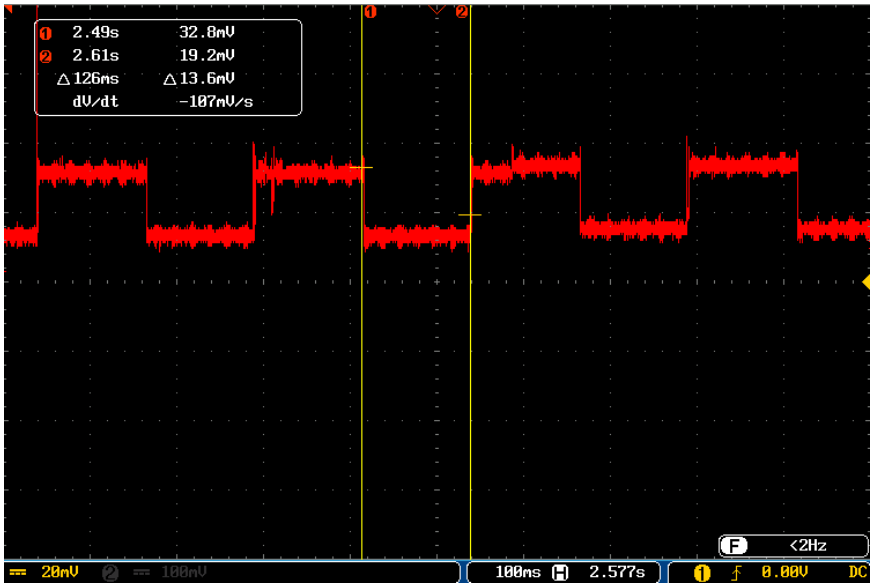

Figure 5.21: Complete frame - Sleep slot

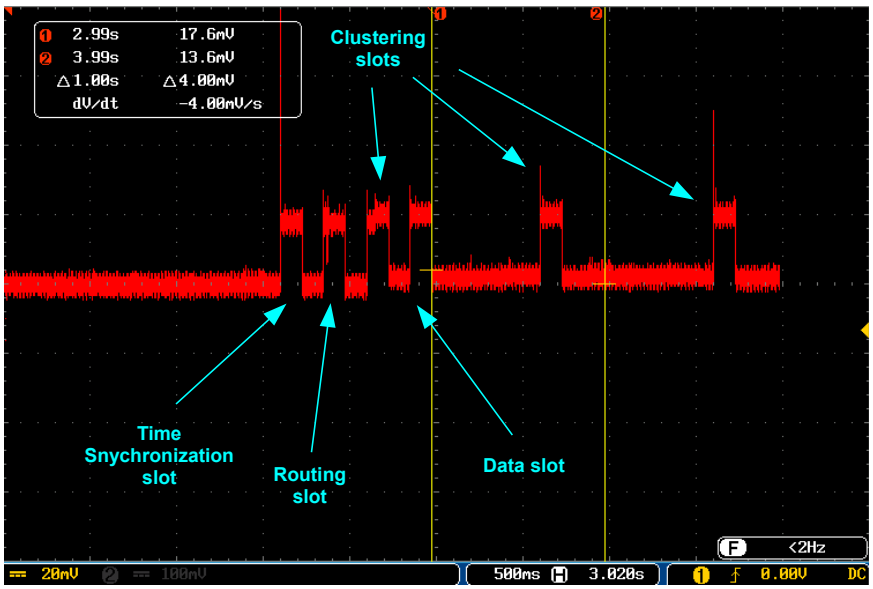

Figure 5.22: Beginning of the cluster formation rounds

- In Figure 5.26 after losing its time synchronization, instead of waiting at least 3 time synchronization slots (which is about 180 seconds), it is observed that node recovers it at about 40 seconds (see Section 4.4).

- In Figure 5.27 it is observed that the sensor node's radio unit is sleeping most of the time in a long synchronized operation as expected, since minimum and

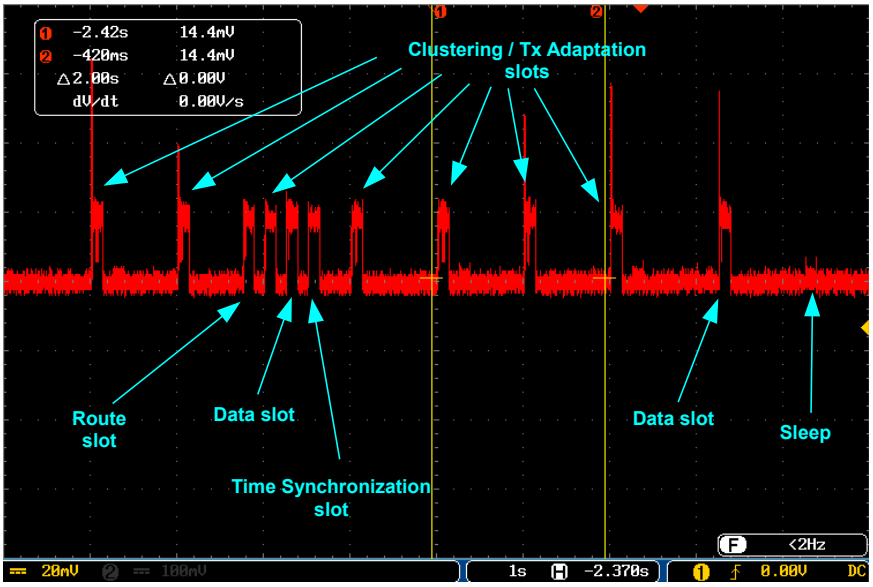

Figure 5.23: Synchronized operation

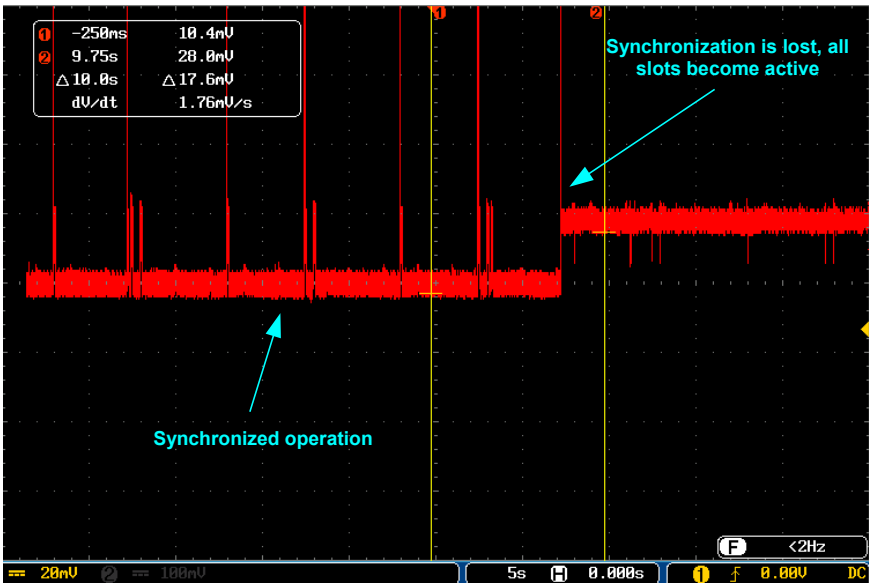

Figure 5.24: Loss of synchronization

maximum duty cycle expectations are stated as approximately 4% in Section 4.5.

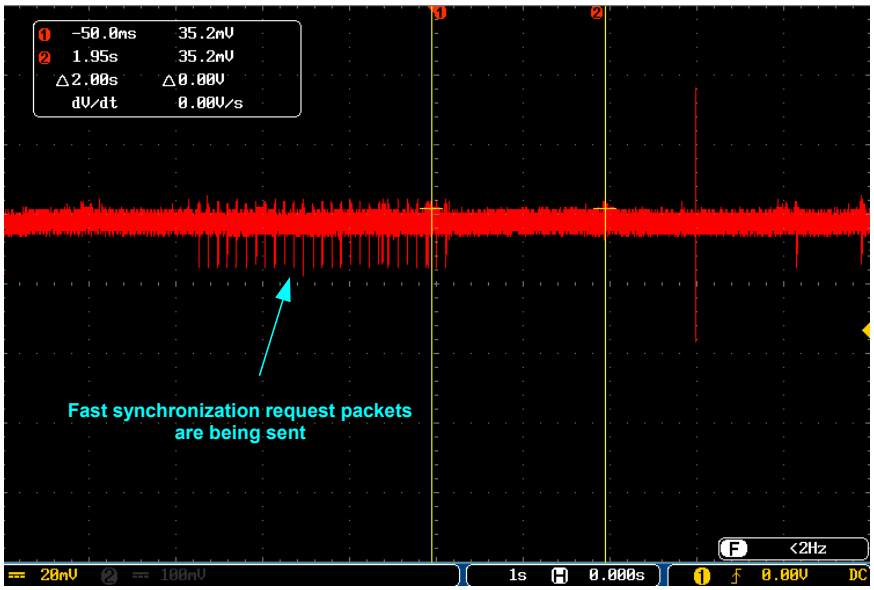

Figure 5.25: Fast synchronization request packets

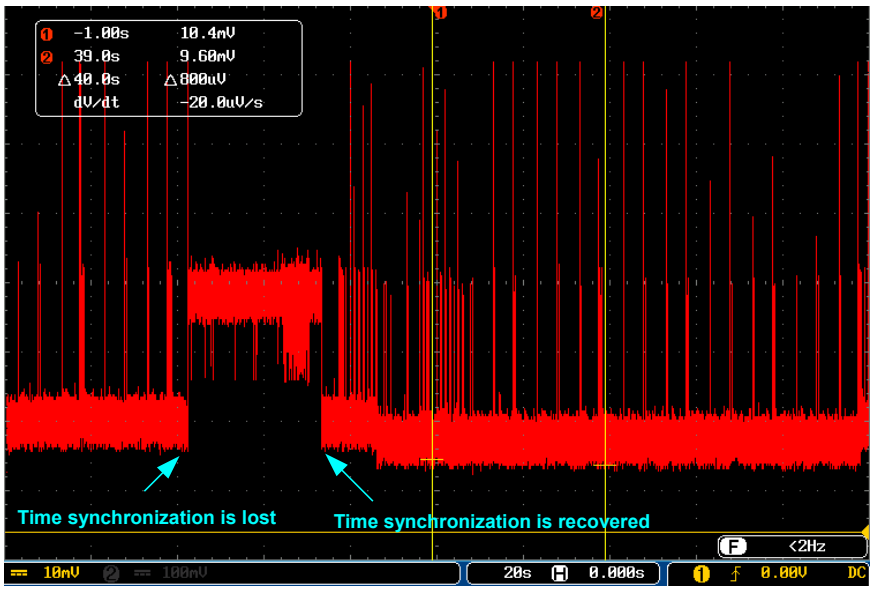

Figure 5.26: Synchronization recovery

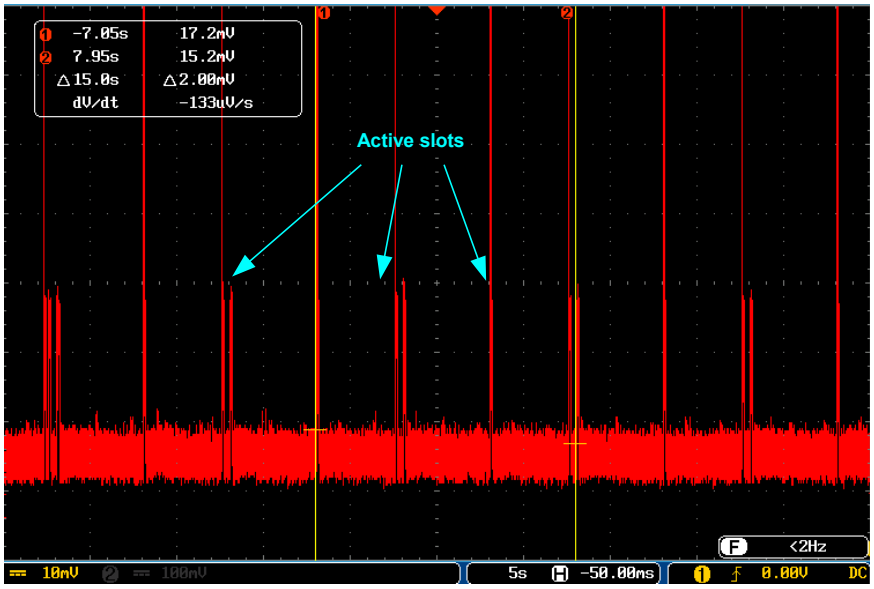

Figure 5.27: Long synchronized operation

5.3 Discussion

Figure 5.4 compares the battery voltages of the nodes during Experiment I. It can be observed that none of the motes' battery voltages drop too quickly. It would be expected the battery voltages for the motes which are frequently elected as cluster heads to diminish faster. However, if both Figure 5.10 and Figure 5.4 are examined carefully, it is seen that being frequently elected as cluster head does not affect the battery voltages significantly. This is due to the fact that the listening costs generate the greatest majority of the total communication related energy costs. This can be seen in Figure 5.5, Figure 5.6, Figure 5.7 and Figure 5.8.

If Figure 5.4 and Figure 5.10 are examined together, it can be seen that Node 5 and Node 7, which have low battery voltages, are not elected as cluster heads. This validates that cluster head elections are based on remaining battery voltages as explained in Section 4.2.

In Figure 5.6 it can be observed that there are minor variations between the nodes' energy consumptions for data packets. As all nodes transmit data packets with a period of 5s as stated in Section 4.6 and they are powered up with about 10s separations in time at startup, the number of data packets transmitted by them are very close. It is also known that data packet sizes for all nodes are the same. Thus, these minor variations are related with the battery voltages of the nodes. It can be observed from Figure 5.4 and Figure 5.6 that the higher the node's battery voltage, the higher the energy consumption for data packets it has.

If Figure 5.6 and Figure 5.7 are examined together, it can be seen that energy consumptions for overhead packet transmissions are slightly greater than energy consumptions for data packet transmissions. This is due to the fact that the data sending rates of the nodes are as low as 0.2 packets per second as stated in Section 2. If data rates were increased, energy consumptions for data packets would be much higher.

In Figure 5.7 it can be observed that base station's energy consumption for transmitting overhead packets is notably greater than the other nodes' consumptions. This is due to the fact that all data packets received by the base station are forwarded to the PC and their costs are accepted as overhead.

If Figure 5.7 and Figure 5.10 are examined together, it can be seen that energy consumptions for overhead packet transmissions are greater for the nodes which are elected as cluster heads at most. This is an expected result, since the energy consumption for forwarding children nodes' data packets is accepted as overhead cost for each cluster head node (see Section 4.7).

In Figure 5.8 it can be observed that energy consumption for sleeping periods are negligible among the other energy costs as expected (see Section 4.7).

Figure 5.11 compares the battery voltages of the nodes during Experiment II. It can be observed that all motes' battery voltages drop quickly compared to Experiment I. It is an expected result, since the motes' radio units are always active during the operation in Experiment II. Additionally, it can be seen that Node 4 and Node 6 deplete their battery voltages quicker than the other nodes. This may be because of their battery quality, since during the experiments MICAz motes were powered by different types of batteries, which are products of different companies.

If Figure 5.5 and Figure 5.12 are examined together, it can be seen that the energy consumption for listening periods in Experiment II is nearly 20-25 times the energy consumption for the same periods in Experiment I for each node. This result is in parallel with the results obtained for the battery voltages. The reason for that is the 100% duty cycles of the radio units of the motes in Experiment II.

Figure 5.14 compares the battery voltages of the nodes during Experiment III. It is again observed that none of the motes' battery voltages drop too quickly. It would be expected for the motes to diminish their battery voltages faster, for the reason that the transmission power adaptation algorithm was disabled in Experiment III. However, if Figure 5.4 and Figure 5.14 are examined together, it can be seen that disabling transmission power adaptation algorithm does not affect the variation in battery voltages significantly. This is due to the fact that the energy consumption for data packet transmissions, in which adapted power levels are used, is nearly 0.3% of the total communication related energy consumption for each node. This fact can be observed in Figure 5.5, Figure 5.6 and Figure 5.7 Energy consumption for sleeping periods can be ignored while considering total communication related energy consumptions herein, since it is observed in Experiment I that they are negligible.

If Figure 5.5 and Figure 5.15 are examined together, it can be seen that the energy consumption for listening periods in Experiment III is slightly less than the energy consumption for the same periods in Experiment I for each node. This result is expected, since motes' CC2420 radio units sleep for the slot durations reserved for the

transmission adaptation packets in Experiment III.

If Figure 5.6 and Figure 5.16 are examined together, it can be seen that the energy consumption for data packets is not as high as expected for each node in Experiment III. On the other hand, it can be seen in Figure 5.4 and Figure 5.14 that battery voltage values are noticeably higher for Experiment I. Thus, high battery voltage values of the motes in Experiment I may be hiding the effect of transmission power adaptation on the energy consumption for data packet transmissions.

If Figure 5.7 and Figure 5.17 are examined together, it can be seen that the energy consumption for overhead packets are in about the same range. Thus, it can be deduced that transmission adaptation packets does not create a significant overhead cost for the implementation as expected, since a maximum of 1/60 packets per second is transmitted for adaptation as explained in Section 4.5.

If Figure 5.9, Figure 5.13 and Figure 5.18 are examined together, it can be seen that motes in Experiment I have the least packet delivery ratios to the base station. As the tranmission power levels decrease, which is a result of the transmission power adaptation algorithm, signal strengths at the receiving nodes also decrease and naturally this lead to packet losses. It can be also seen that in Experiment II and III, almost the same ratio of the packets are delivered. This validates that the sleep scheduling of the radio unit works properly in the small network deployed for the experiments. Thus, the reason for increased packet loss rates can be the adapted transmission power levels of the motes.

Figure 5.19 shows the transmission power levels selected by Node 6 during Experiment I and III. It can be seen that the adapted power levels are often below the non-adapted power levels as explained in Section 4.3.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis work, a WSN implementation with many features has been realized. HEED clustering algorithm has been implemented on a testbed. For the time being, TinyOS 2 implementation of HEED is not available in neither TinyOS 2 contribution directories nor any open source repositories. TinyOS 2 implementation of HEED will be released for the community access. HEED implementation has been integrated with sleep scheduling. In order to make sleep scheduling work smoothly, a time synchronization mechanism, which also provided the accuracy of the timestamps of the sensor samples, has been implemented. Thus, the output of this thesis is a system with many features. It may be deployed in real world, if its limitations are overcome. Furthermore, the infrastructure of the system is valuable for allowing researchers to work on popular issues such as routing, clustering, time synchronization and sleep scheduling in WSNs.

In order to validate the implementation, energy consumption calculations have been made and presented as experimental results. Researchers often focus on transmission related energy consumption and omit idle listening energy consumption of the radio units while studying on energy efficiency in WSNs. In this thesis, a detailed, real-world energy consumption analysis, which did not omit idle listening, even sleeping energy consumption, in a network composed of MICAz motes is provided.

However, the implementation has some limitations that needs to be improved.:

- The sensor nodes were assumed to be non-mobile. Thus, mobility issue for the implementation is open for research.
- Duration of the time slot reserved for data packet transmissions was not optimized. In periodic data sampling and forwarding approach, duration of the data slot can be shorter for cluster member nodes, since they do not expect any packets from neighbor nodes after sending its data packet. This kind of a mechanism, will obviously extend the network lifetime. Since the duty cycles of the member nodes will be less than the duty cycles of the cluster head nodes, the cluster head nodes will begin to consume more energy. Thus, HEED performance using battery constraint will be highlighted better. In triggered approach, duration of the data slot can be adaptive for cluster heads. Depending on the sensing activities of the member nodes, cluster heads may adapt duration of their data slot. However, this may introduce problems of scalability. The children nodes in depths of the network tree may never be able to reach base station, if they

miss the active duration of the nodes closer to the base station due to multi-hop delays.

- Duration of the clustering and routing slots can be shorter, since multi-hop packets are not being transmitted in those slots. This may reduce overall energy consumption for all motes either a cluster head or a member node, since the duty cycles of all will decrease significantly.

- Received Signal Strength Indicator (RSSI) values may be used in transmission power adaptation algorithm. Additionally, other transmission power adaptation algorithms in literature may be investigated. However, they must be carefully analyzed whether they can fit into the implementation or not. Since the transmission power adaptation scheme severely effects cluster sizes, this may produce incompatibilities. These incompatibilities may damage the stable network operation, which may result in worse overall energy consumption and packet loss ratings. In the implementation, the best-effort multi-hop routing protocol of TinyOS 1 was ported to TinyOS 2 for constructing routes towards the base station among cluster head nodes. A better routing protocol can be used. However, a heavy-weight routing protocol will not be applicable, since MICAz motes have a serious memory limitation. It will be feasible to use more advanced motes, if the multi-hop routing algorithm in the implementation is to be replaced.

- Data aggregation techniques, which may significantly improve energy efficiency in WSNs by reducing the number of transmitted packets, were out of the scope of the implementation. They can be researched and implemented.

- The implementation was tested on a small testbed composed of 12 MICAz motes at maximum. It can be tested on a larger testbed for scalability.

- A modified version of HEED clustering protocol is realized in the implementation. The other clustering protocols can be realized. Performances of them can be compared and new protocol proposals can be made using the infrastructure provided in the implementation.

REFERENCES

[1] A. Moshaddique, I. SM Riazul, and K. Kyungsup, "Energy Saving Mechanisms For MAC Protocols In Wireless Sensor Networks," *International Journal of Distributed Sensor Networks*, 2010.

[2] A. Bereketli and S. Bilgen, "Remotely Powered Underwater Acoustic Sensor Networks," *Sensors Journal, IEEE*, vol. 12, no. 12, pp. 3467–3472, 2012.

[3] J. Al-Karaki and A. Kamal, "Routing Techniques In Wireless Sensor Networks: A Survey," *Wireless Communications, IEEE*, vol. 11, no. 6, pp. 6–28, 2004.

[4] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An Application-Specific Protocol Architecture For Wireless Microsensor Networks," *Wireless Communications, IEEE Transactions on*, vol. 1, no. 4, pp. 660–670, 2002.

[5] X. Liu, "A Survey On Clustering Routing Protocols In Wireless Sensor Networks," *Sensors*, vol. 12, no. 8, pp. 11113–11153, 2012.

[6] S. Lindsey, C. Raghavendra, and K. M. Sivalingam, "Data Gathering Algorithms In Sensor Networks Using Energy Metrics," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 9, pp. 924–935, 2002.

[7] M. Ye, C. Li, G. Chen, and J. Wu, "EECS: An Energy Efficient Clustering Scheme In Wireless Sensor Networks," in *Performance, Computing, and Communications Conference, 2005. IPCCC 2005. 24th IEEE International*, pp. 535–540, 2005.

[8] C. Li, M. Ye, G. Chen, and J. Wu, "An Energy-Efficient Unequal Clustering Mechanism For Wireless Sensor Networks," in *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, pp. 3–17, IEEE, 2005.

[9] O. Younis and S. Fahmy, "HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach For Ad Hoc Sensor Networks," *Mobile Computing, IEEE Transactions on*, vol. 3, no. 4, pp. 366–379, 2004.

[10] J. Ali, G. Kumar, and M. K. Rai, "Major Energy Efficient Routing Schemes In Wireless Sensor Networks: Survey And Idea," *International Journal of Computers and Technology*, vol. 4, no. 2, pp. 261–266, 2013.

[11] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time Synchronization Using Reference Broadcasts," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 147–163, 2002.

[12] M. Roche, "Time Synchronization In Wireless Networks," *CSE574S: Advanced Topics in Networking: Wireless and Mobile Networking (Spring 2006), Washington University in St. Louis*, vol. 23, 2006.

[13] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync Protocol For Sensor Networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 138–149, ACM, 2003.

[14] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The Flooding Time Synchronization Protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 39–49, ACM, 2004.

[15] C. Lenzen, P. Sommer, and R. Wattenhofer, "Optimal Clock Synchronization In Networks," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pp. 225–238, ACM, 2009.

[16] P. Levis and D. Gay, *TinyOS Programming*. Cambridge, UK: Cambridge University Press, 2009.

[17] P. Levis, "TinyOS Enhancement Proposal 111: message_t." `http://www.tinyos.net/tinyos-2.1.0/doc/html/tep111.html`, Aug. 2013.

[18] D. Gay, P. Levis, D. Culler, and E. Brewer, *NesC 1.1 Language Reference Manual*, May 2003.

[19] "TinyOS Documentation Wiki." `http://tinyos.stanford.edu/tinyos-wiki/index.php/Main_Page`, Aug. 2013.

[20] Crossbow Technology, Inc, *MPR-MIB Users Manual*, June 2006.

[21] Chipcon Products from Texas Instruments, *CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*, Apr. 2006.

[22] Crossbow Technology, Inc, *MTS/MDA Sensor Board Users Manual*, June 2006.

[23] TinyOS Working Group, "Getting Started with TinyOS." `http://tinyos.stanford.edu/tinyos-wiki/index.php/Getting_Started_with_TinyOS`, Aug. 2013.

[24] Crossbow Technology, Inc, *Wireless Sensor Networks, Getting Started Guide*, Sept. 2005.

[25] L. Wang, *Topology-Based Routing for Xmesh in Dense Wireless Sensor Networks*. ProQuest, 2007.

[26] A. Woo, T. Tong, and D. Culler, "Taming the Underlying Challenges Of reliable Multihop Routing In Sensor Networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pp. 14–27, ACM, 2003.

[27] T. Azim and P. Levis, "Tinyos Release Notes: Porting TinyOS 1.x Code to TinyOS 2.0." `http://www.tinyos.net/tinyos-2.x/doc/html/porting.html`, Aug. 2013.

[28] O. Younis and S. Fahmy, "An Experimental Study Of Routing And Data Aggregation In Sensor Networks," in *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, pp. 50–57, 2005.

[29] M. Maroti and J. Sallai, "TinyOS Enhancement Proposal 132: Packet Timestamping." `http://www.tinyos.net/tinyos-2.1.0/doc/html/tep132.html`, Aug. 2013.

[30] C. Sharp, M. Turon, and D. Gay, "TinyOS Enhancement Proposal 102: Timers." `http://www.tinyos.net/tinyos-2.1.0/doc/html/tep102.html`, Aug. 2013.

[31] J. Ma, W. Lou, Y. Wu, X. Y. Li, and G. Chen, "Energy Efficient TDMA Sleep Scheduling In Wireless Sensor Networks," in *INFOCOM 2009, IEEE*, pp. 630–638, IEEE, 2009.

[32] D. Moss, J. Hui, P. Levis, and J. Choi, "TinyOS Enhancement Proposal 126: CC2420 Radio Stack." `http://www.tinyos.net/tinyos-2.1.0/doc/html/tep126.html`, Aug. 2013.

[33] J. O'Madadhain, D. Fisher, P. Smyth, S. White, and Y.-B. Boey, "Analysis And Visualization Of Network Data Using JUNG," *Journal of Statistical Software*, vol. 10, no. 2, pp. 1–35, 2005.

[34] J. Gosling, B. Joy, G. Steele, and G. Bracha, *Java (TM) Language Specification, The (Java (Addison-Wesley))*. Addison-Wesley Professional, 2005.