LECTES: A Low Energy Consumption Time Estimation Scheme for WSNs

Kenji Yonekawa

Faculty of Environment and Information Studies

Keio University

5322 Endo Fujisawa Kanagawa 252-8520 JAPAN

Submitted in partial fulfillment of the requirements for the degree of Bachelor

Advisors:

Professor Hideyuki Tokuda
Professor Jun Murai
Associate Professor Hiroyuki Kusumoto
Professor Osamu Nakamura
Associate Professor Kazunori Takashio
Assistant Professor Noriyuki Shigechika
Assistant Professor Rodney D. Van Meter III
Associate Professor Keisuke Uehara
Associate Professor Jin Mitsugi
Lecturer Jin Nakazawa
Professor Keiji Takeda

Abstract of Bachelor's Thesis

LECTES: A Low Energy Consumption Time Estimation Scheme for WSNs

Advances in micro electro-mechanical systems (MEMS) and wireless technologies have lead to the development of small, low power sensor nodes with wireless communication ability. Researchers have been proposing and implementing applications that use wireless sensor networks (WSNs), which consist of hundreds of wireless sensor nodes. In most of these applications, sink nodes are required to know the time when data are sensed, thus, sensor nodes in WSNs are desired to have accurate clocks. However, due to the strict constrain of resources, the clocks of wireless sensor nodes tick at different rate. Many researchers have been proposing time synchronization protocols, which synchronize sensor nodes' clocks. Although they achieved high accuracy, they consume large amount of energy, and lack scalability.

This thesis proposes a time estimation scheme called LECTES, which is a replacement of time synchronization protocols. LECTES is designed to operate with considerably small amount of energy compared to time synchronization protocols. We implement LECTES on mote, which is the most widely used wireless sensor node. In order to evaluate LECTES, we compare the energy consumption, accuracy, scalability and storage usage against those of simple WSN application which sends sensed data at a specific frequency, and FTSP which maintains high accuracy with small amount of energy. The evaluation results prove that LECTES is superior to FTSP in terms of energy consumption, accuracy, scalability and storage usage. Even though LECTES achieves high scalability, we still need to improve it, so that it can be adapted to large scale WSNs.

Kenji Yonekawa

Faculty of Environment and Information Studies, Keio University

卒業論文要旨 2009年度 (平成21年度)

LECTES:無線センサネットワークにおける
低消費電力な時刻推測機構の設計と実装

近年，MEMS (Micro Electro-Mechanical Systems) の発達に伴い，センサノードは小型化，安価化されている．また，無線技術の発達により，無線通信機能を備えた小型センサノードが普及してきている．これにより，無線センサノードでネットワーク (WSN:Wireless Sensor Network) を構築し，それを用いるアプリケーションが数多く提案，実装されている．WSN アプリケーションの多くでは，sink ノードがデータを解析する際に，それが取得された時刻を知る必要がある．一方，無線センサノードは小型化のトレードオフとして，計算能力やストレージ，バッテリなどが厳しく制限されている．また，CPU やオシレータの精度が低いことから，無線センサノードの保持するクロックが実際の時刻からずれていくことが分かっており，多くのアプリケーションにおいてデータを処理する際のエラーの原因となっている．

上記の問題を解決するために，WSN のための時刻同期が提案されてきた．WSN 内の各ノードの時刻を同期することにより，時刻のずれの問題は解消されたが，消費電力の大きさ，スケーラビリティ，クロックの修正方法などの問題が残っている．本研究では，時刻同期に変わる時刻推測手法，LECTES (Low Energy Consumption Time Estimation Scheme) を提案，実装する．LECTES は既存の時刻同期と比較し，電力消費を減らすことを主眼としており，実用に耐えうる時刻の精度やスケーラビリティを確保することを目的とする．本機構を無線センサノードの中で，もっとも利用頻度の高い mote 上に実装した．評価は，データをセンシングし，その送信のみを行う WSN アプリケーションと，精度が高く電力消費の少ない FTSP (Flooding Time Synchronization Protocol) と LECTES において，電力消費，精度，スケーラビリティ，使用データ量について比較，考察した．評価結果から，比較した 4 つの項目全てにおいて，LECTES が FTSP よりも優れている事が実証された．今後の展望として，巨大なマルチホップ WSN に対応できるよう，スケーラビリティを改善する．

慶應義塾大学　環境情報学部

米川 賢治

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Recent advances in micro electro-mechanical system (MEMS) have lead to the miniaturization of sensor nodes. In addition, growth of wireless technologies provides wireless communication ability to the sensor nodes. As a result, many researchers have been proposing and implementing applications that use a wireless sensor network (WSN), which consists of hundreds of wireless sensor nodes.

Since wireless sensor nodes are made to be as small as possible, they have strict restriction on resources, such as CPU, memory, storage and energy source. Due to the severe constrain, wireless sensor nodes have low precision CPU and oscillator, which results in poor clock accuracy. At the same time, many WSN applications require wireless sensor nodes to have certain precision of time in order to work correctly.

Network Time Protocol (NTP) [21] is most widely used time synchronization protocol for computers connected to the Internet. However, NTP is not accurate enough for synchronizing wireless sensor nodes' clocks, because they demand appreciably high accuracy. Furthermore, NTP is designed for resource-rich nodes, therefore, it eats up large amount of energy. Requiring large amount of energy is unfavorable for wireless sensor nodes, since they have severe limitation on energy source. Hence, researchers have been proposing protocols

that synchronizes sensor nodes' clocks.

1.2 Challenges and Goal

We have to keep in mind that resources of wireless sensor nodes are extremely limited. Hence, we propose four requirements for solving time precision problem in WSNs as follows: 1) energy consumption, 2) accuracy, 3) scalability, and 4) storage usage. In other words, we want a solution that requires small amount of energy, while achieving high accuracy and scalability with minimum storage usage. In case of personal computers, electronic power is always provided. On the other hand, most wireless sensor nodes are designed to run with batteries mounted on them. Therefore, one of the most important characteristics of wireless sensor nodes is that they have strictly limited energy source, and thus, our primary goal is to minimize the energy usage as much as possible.

Another problem of sensor nodes concerning time precision problem is their low performance CPU and low precision oscillator. Because of them, wireless sensor nodes' clocks drift from the actual clock much quicker than those of personal computers. This, in turn, shows that in order to maintain the differences between sensor nodes' clocks within certain value, time synchronization have to take place quite frequently. Even though the energy consumed by single cycle of time synchronization is small, large amount of energy is required when the frequency of synchronization is high. In other words, no matter how time synchronization methods suppress the energy usage, they have to be invoked frequently especially for wireless sensor nodes, which ends up in consuming large amount of energy. Therefore, along with minimizing the energy consumption, frequency of time synchronization must be minimized or eliminated.

802.15.4 is most widely used wireless personal area network (WPAN) technology for wireless sensor nodes, whose speed, cost and energy usage is considerably low compared to other

wireless technologies. Due to these characteristics, 802.15.4 is fragile against interferences. Current protocols synchronize sensor nodes' clocks by exchanging messages among sensor nodes, which has a high chance of causing packet interference. Since packet loss rate is better to be low, we need to minimize or eliminate the number of message exchange among sensor nodes.

We propose Low Energy Consumption Time Estimation Scheme (LECTES) as a replacement of time synchronization protocols. LECTES is designed to run with considerably small amount of energy, while achieving high accuracy. LECTES assures high scalability and small storage usage as well.

1.3 Structure of Thesis

This thesis is organized as follows. In Chapter 2, we describe the time precision problem in WSNs with its background, relation to WSN applications and related work. We analyze the time precision problem in WSNs by proposing our target environment with its requirements and discuss the problems of time synchronization protocols in Chapter 3. In Chapter 4, we propose LECTES,and explain its characteristic and feature. We then present the design and implementation of LECTES in Chapter 5. Chapter 6 provides the purpose, methodology, comparison targets and evaluation results of LECTES. Finally, in Chapter 7, we conclude this thesis and introduce our future work.

Chapter 2

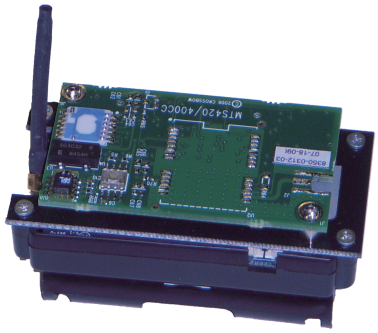
Time Precision in WSNs

2.1 Introduction

This chapter introduces the time precision problem in WSNs. First, we describe the background of time precision problem. We then present the actual WSN applications assuming certain precision of time. Finally, we introduce related work, and summarize this chapter.

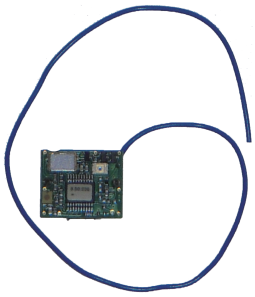
2.2 Background

For the past few decades, the advances in technologies relating to MEMS, electronic circuits' integration/packaging, various kinds of sensors, and batteries have lead to the development of tiny nodes attached with sensors called sensor nodes. The advances in these technologies allowed sensor nodes to be smaller and cheaper. In addition, progress in wireless technologies provides these sensor nodes the ability to communicate with others. These sensor nodes with wireless communication abilities are called wireless sensor nodes, whose examples are mote [7] (Fig. 2.1(a)), SunSpot [20] (Fig. 2.1(b)), and μpart [5] (Fig. 2.1(c)). Improvement of wireless technologies are essential for lengthening sensor nodes' lifetime, since wireless communication consumes large amount of energy. One of the wireless specifications, ZigBee plays a significant role in extending wireless sensor nodes' lifetime. ZigBee is specialized to wireless sensor nodes, and used to create short range ad-hoc networks with low energy and



(a) Mote
58mm x 32mm x 7mm
runs with two AA batteries

(b) SunSpot
63.5mm x 38.1mm x 12.7mm
runs with LI-ION prismatic battery



(c) μpart
20mm x 17mm x 7mm
runs with one CR1632 battery.

Figure 2.1: Sensor Nodes

low power.

Since one of the most notable characteristics of sensor nodes is their size, their performance is sacrificed for it. Furthermore, since most wireless sensor nodes run with batteries, chips and modules for them are made to consume small amount of energy, which often are low performance.

2.3 WSN Applications and Precision of Time

Researchers have been proposing applications which use WSNs [12] [11] [29] [28] [18] [1]. In these applications, sink nodes, which usually are high performance nodes, collect and analyze data transmitted by the sensor nodes. Clock drifting of sensor nodes is one of the most critical problems in WSNs, since it has a tremendous effect when analyzing data, and may end up in unreliable results.

Gyula et al. proposed a counter sniper application [12], which locates the position of a sniper using WSNs. By having multiple sensor nodes detect the movement of bullet, they successfully locate the position of sniper in three dimensional urban environment much ac-

curately and quickly compared to the counter sniper application which uses infrared cameras and microphones [17]. When using WSN to locate the position of sniper, the differences in sensor nodes' clocks must be less than 1 microsecond, otherwise, for every 1 microsecond jitter, the estimated location of sniper differs up to 1 meter. Target tracking application [11] uses the information obtained from sensor nodes placed in the environment, and by processing them, it approximates the position of a target at given time. This application demands the precision of at least 100 microseconds in order to process data correctly, however, the required precision depends on the speed of targets. Target tracking application may be used to detect the collision of humans and/or vehicles, hence, mistakenly estimating the targets' position could result in accidents. Xu et al. suggest that, by placing numbers of sensor nodes in structures and acquiring the acceleration of them, the healthiness of structures can be judged [29]. For reliable judgement of structures' healthiness, the differences between sensor nodes' clocks must be less than 250 microseconds. Corruption of structures can lead to serious tragedy, thus, the precision of sensor nodes' clocks must be maintained with care. Since it is dangerous for human to go near the volcanoes, application that monitors the conditions of them without human being's help is essential. Werner-Allen et al. introduced an application that monitors the state of volcano using WSNs [28]. This application expects sensor nodes' clocks to have jitter less than 10 milliseconds. Miss-evaluation of volcano monitoring could lead to threatening human lives, or making them evacuate for no reason.

WSN applications require sensor nodes' clocks to have jitter within certain value. Permissible jitter differs for each application, which is usually microsecond to millisecond order. However, due to wireless sensor nodes' characteristics, their clocks drift from the actual clock much quicker than those of high performance nodes, such as personal computers. It is known that clock of mote [7], which is most widely used wireless sensor nodes, drifts from the actual clocks at most 40 microseconds per second. This is because, they have low performance

CPU and oscillator. In WSN with multiple motes, the maximum jitter between sensor nodes' clocks may increase 80 microseconds per second in the worst case, which messes up a lot of WSN applications. For these reasons, it is important for sink nodes to know the accurate time when date are acquired. Researchers have been proposing methods to synchronize sensor nodes' clocks in WSN as solutions for time precision problem [9] [10] [22] [19] [24] [30].

2.4 Related Work

In this section, we introduce three protocols that solve the time precision problem; RBS, CCS and FTSP. Their approach, characteristics and achievements are described in detail.

2.4.1 RBS: Reference Broadcast Synchronization

Elson et al. proposed a time synchronization protocol called Reference Broadcast Synchronization (RBS) [9]. The author claims that RBS achieved the precision of 11 microseconds on Mica mote. However, authors of Timing-sync Protocol for Sensor Networks (TPSN) [10] performed an experiment using TPSN and RBS, which resulted in 16.9 and 29.1 microseconds jitter in a single hop network, respectively. Even though TPSN achieves higher precision than RBS, TPSN cannot be used in platforms except for Mica platform, because of its implementation. Hence, we do not compare TPSN against our scheme.

RBS is implemented in application layer, and therefore, it can be adapted to any kind of platform by implementing its protocol. When synchronizing clocks, RBS reduces the effects of jitter by omitting the error factors relating to sending. In RBS, every sensor nodes in a WSN broadcast a packet at specific frequency. Sensor nodes which received the packet exchange messages with neighbor nodes, which also received the same packet. RBS does not use the sender's time as a reference, instead, it uses the reception time of packets on receiver nodes as a reference. By doing so, RBS eliminates the error relating to transmission.

When synchronizing sensor nodes' clocks, it is important to reduce the effects of the error factors. There are three major elements which cause jitter in time when exchanging packets: send, propagate and receive. RBS improves the accuracy by removing the effects of jitters relating to sending.

2.4.2 CCS: Continuous Clock Synchronization

Mock et al. proposed Continuous Clock Synchronization (CCS) [22]. CCS achieves the precision of 140 microseconds, which is substantially low compared to that of RBS [9], TPSN [10] and FTSP [19], however, its idea is notable. The authors of CCS stated that instantaneous clock synchronization is not suitable for WSNs. They suggest that, by synchronizing clocks continuously, wireless sensor nodes can maintain their clocks with high accuracy. Clocks of Wireless sensor nodes drift from the actual clocks much quicker than those of personal computers. Thus, the clocks of wireless sensor nodes must be synchronized frequently in order to maintain certain precision. CCS came up with the idea of "continuous" clock synchronization for wireless sensor nodes, which allows to uphold their clocks within certain precision at any given time.

Another notable characteristic of CCS is the existence of virtual clock. The author of CCS claims that modifying the clocks of sensor nodes has tremendous overhead in terms of calculation and energy consumption. Moreover, when modifying sensor nodes' clocks, there is a chance of skipping or redoing scheduled tasks, which most application does not allow. By using virtual clocks, sensor nodes do not have to modify their actual clocks any longer. Moreover, there is no worry about skipping or redoing tasks. Virtual clocks can be easily modified with low overhead. In addition, CCS is strong against packet interference, because, sensor nodes can make their virtual clocks tick at their preferred rate, which may suppress the jitter of clocks even with some information missing.

2.4.3 FTSP: Flooding Time Synchronization Protocol

The Flooding Time Synchronization Protocol (FTSP) [19] synchronizes sensor nodes' clocks extremely accurately. According to their experiment, the average error of FTSP in a single hop network is 1.48 microseconds. Wireless sensor nodes using FTSP synchronize clocks according to a flooding model. FTSP does not assume the existence of an accurate clock source, instead, it synchronizes "root" node's clock to every other nodes in WSNs, and root node is dynamically, periodically and randomly chosen among the sensor nodes in WSNs. Root node broadcasts its time, and sensor nodes which received that packet adjusts their clocks according to it. After adjusting clocks, they broadcast their time, and so on, which is how FTSP provides multi hop time synchronization.

FTSP is implemented in MAC layer of motes. FTSP improved the accuracy by reducing the effect of send and receive jitter by implementing in low layer. FTSP's implementation is platform specific, because it is implemented in MAC layer of mote, and cannot be used in other platforms whose MAC layer is not implemented with software.

2.5 Summary

In this section, we summarize this chapter. The advances in MEMS and wireless technologies allowed the development of wireless sensor nodes. The improvement of technologies results in wireless sensor nodes to be smaller and cheaper, which leads to the spreading of WSN, a network consisting of hundreds of wireless sensor nodes. Because of the severe resource constrains, clocks of sensor nodes drift from the actual clocks quickly, which causes a lot of WSN applications to work incorrectly. In order to solve the time precision problem in WSNs, researchers have been proposing protocols that synchronize sensor nodes' clocks in WSNs [9] [10] [22] [19] [24] [30].

Chapter 3

Analysis of Time Precision Problem

3.1 Introduction

In this chapter, we analyze the time precision problem in WSNs. First, we describe the environment we are targeting. Then, four requirements of the target environment is stated: low energy consumption, high accuracy, high scalability and small storage usage. Third, Problems of time synchronization are indicated. Finally, we summarize this chapter.

3.2 Target Environment

In this section, we describe our target environment. We assume two factors; large scale WSNs and the existence of sink node. Firstly, we discuss the validity of targeting large scale WSNs. Then, we address the adequacy of assuming the existence of sink node in WSNs.

3.2.1 Large Scale WSN

There are many kinds of WSN applications, and each of them assumes different kind of environment. The size of the WSN depends on the target and goal of the application. The bigger the size of WSN is, the more it gets affected by the time precision problem, especially in multi hop WSNs, because the effect of jitter between sensor nodes' clocks would be accumulated. In large scale WSNs, sensor nodes consume greater amount of energy compared

Table 3.1: Energy Consumption of Modules in Micaz and Iris Mote

	Micaz	Iris
CPU Active	12 mA	10 mA
CPU Idle	8 mA	2.2 mA
TX (max)	17.4 mA	16.5 mA
TX (min)	8.5 mA	9.5 mA
RX	18.8 mA	15.8 mA

to those in small scale WSNs, because they transmit, receive and forward greater number of packets.

Table 3.1 shows the energy consumption of CPU being active/idling, and radio chip sending/receiving packets. The data in the table is based on data sheet of CPU and radio chip of Micaz and Iris mote [3] [2] [14] [4]. It is proven that transmitting packets demand about 1.45 and 1.65 times more energy compared to CPU being active in Micaz and Iris mote, respectively. Moreover, receiving packets require as much energy as sending them. This demonstrates that the use of radio demands huge amount of energy compared to other operations, such as calculation and sensing.

For these reasons, we target applications using large scale WSN. By targeting more difficult environment, our scheme can be adapted not only to large scale WSNs, but also to small scale WSNs. Hence, it is reasonable to target for large scale WSNs. We peculiarly need to assure low energy usage, accuracy, scalability and robustness for large scale WSNs.

3.2.2 Existence of Sink Nodes

We suppose WSN applications to have at least one sink node. WSN applications can be divided into two types: 1) WSNs that have at least one sink node, which analyzes data transmitted by other sensor nodes, and 2) WSNs that have no sink node, and data are stored on each sensor node. Nearly all applications are using the former type of WSNs.

Having small storage is one of the characteristics of wireless sensor nodes, and because of it, storing data on them is not realistic. One way of achieving latter type is to equip sensor nodes with large storage, however, the energy consumption would increase, which is unenviable.

Since the majority of WSNs has at least one sink node, assuming the existence of sink node in WSNs is proper. Therefore, we aim to solve the time precision problem in WSNs that has at least one sink node.

3.3 Requirements

This section introduces the requirements for the solutions of time precision problem in the target environment. The four requirements, low energy consumption, high accuracy, high scalability and small storage usage are explained in detail as follows.

3.3.1 Low Energy Consumption

Most wireless sensor nodes are equipped with batteries so that they can be placed anywhere. Wireless sensor nodes are usually made to be small, hence, they usually run with small batteries, which results in them having severely limited energy source. For this reason, we should suppress the energy consumption in order to maximize the wireless sensor nodes' lifetimes.

In WSN applications, wireless sensor nodes are usually used to collect data, such as temperature, illuminance, acceleration and humidity. After they gather data using sensors attached to them, they send packets containing the data toward the sink nodes. Since their jobs are sensing and transmitting packets towards the sink nodes, their energy should be used only for that purpose, otherwise, their lifetime would be shorter. Solving the time precision problem is important, although, we should minimize the energy consumption so that the sensor nodes can last long. For these reasons, minimizing the energy consumption is the most important

requirement when solving the time precision problem, since sensor nodes have extremely limited energy.

3.3.2 High Accuracy

Majority of WSN applications require sink nodes to know the time when data are acquired with certain accuracy. The demanded accuracy of time differs for each application, for example, counter sniper application requires the precision of 1 microsecond [12], while volcano monitoring application requires the precision of 10 milliseconds [28]. A scheme is desired to achieve high accuracy, so that it can be adapted to greater number of WSN applications. Most important prerequisites when solving the time precision problem is to minimize the energy consumption. Thus, we aim to implement a scheme which achieves high precision with the use of small amount of energy.

3.3.3 High Scalability

Our target is large scale WSNs, hence, the scalability of our scheme must be high. In order for WSNs to have high scalability, they must support multi hop. However, current schemes that solve time precision problems do not have high scalability, because they have poor accuracy in multi hop networks. This is because, the error in single hop network becomes greater in multi hop network by being accumulated. Therefore, in order to guarantee the scalability, we need to improve the accuracy in multi hop networks.

Another factor that affects the scalability of WSNs is message exchange. Since most sensor nodes are equipped with low power wireless chips, they usually use ZigBee to communicate with each other. ZigBee runs with considerably small amount of energy, however, as a trade off, it loses large number of packets compared to the high power wireless technologies, such as 3G and Wi-Fi. The packet loss rate becomes high when greater number of packet collides

with each other, which occurs when greater number of packets are being exchanged. This, in turn, shows that minimizing the number of message exchange has huge effect on improving the scalability especially in large scale WSNs, because sensor nodes in them exchange large number of packets by default. In order to achieve high scalability, we have to accomplish high accuracy in multi hop WSNs, while minimizing or eliminating the message exchange.

3.3.4 Small Storage Usage

One of the characteristics of wireless sensor nodes is having small storage. In most WSN applications, the only node that has large storage is sink node if any does. Therefore, storing data on wireless sensor nodes is not realistic because they cannot hold large data on them. In fact, minimizing the storage usage of wireless sensor nodes is important in any kind of operation, such as sensing, communicating and using database. Given this characteristic, schemes for solving time precision problem should minimize the storage usage as much as possible under any circumstances.

3.4 Problems of Time Synchronization

In this section, we describe the problems of time synchronization. We describe how current time synchronization protocols do not fulfill the requirements other than accuracy, by stating their problems.

3.4.1 Energy Consumption

One of the biggest problems of current time synchronization protocols is that they consume large amount of energy. Since wireless sensor nodes have strictly constrained energy source, minimizing the energy usage is extremely important. Current time synchronization protocols consume large amount of energy for two reasons as follows.

- Message exchange

  Time synchronization protocols require to exchange messages wireless communication among sensor nodes. Although wireless communication is useful, it eats up large amount of energy. Shnayder et al. measured the energy usage of each operation on Mica2 mote using oscilloscope [25]. They conclude that the energy used by radio transmission is about 2.7 and 6.7 times more compared to that of CPU being active and idle, respectively. Furthermore, the amount of energy consumed by radio reception is slightly smaller compared to CPU being active. Their experiment proves that wireless communication requires tremendous amount of energy.

  If the time synchronization information can be stored in data packets, which are already being exchanged in WSN applications, time synchronization protocols do not have to exchange extra messages for synchronization. However, current time synchronization protocols require to exchange messages other than data messages, because, they have to create their own network to assure that every sensor nodes in WSNs are synchronized. For these reasons, current time synchronization protocols consume large amount of energy, and it is inevitable.

- Frequency of message exchange

  Time synchronization have to take place at certain frequency in order to achieve high accuracy. This is because, clocks of wireless sensor nodes drift from the actual clock very quickly. It is known that clocks of Micaz mote drift from the actual clock at 40 microseconds per second at maximum, which means that the possible maximum increase of jitter in a WSN is 80 microseconds per second in the worst case. Given this fact, even if there is a protocol that can achieve synchronization with no error, if an application demands 100 microseconds precision, time synchronization has to take

place at least every 1.25 seconds. This shows that time synchronization must take place frequently, which results in consuming large amount of energy.

3.4.2 Scalability

The primary goal of current time synchronization protocols is to achieve high accuracy, yet, in order to be used in WSN applications, it is important to ensure high scalability. Number of message exchange affects the scalability; the more the number of message exchange required by time synchronization protocols, the less scalable the WSN becomes. Therefore, in order for time synchronization protocols to have high scalability, they have to minimize or eliminate the number of message exchange. However, in order to ensure synchronization of every sensor nodes in WSNs, current time synchronization protocols cannot eliminate message exchanging. Moreover, some protocols require more than one way communication; RBS [9] requires more than two way communication in each synchronization cycle, which results in them having extremely poor scalability.

3.4.3 Storage Usage

The third problem of current time synchronization protocols is storage usage. A lot of time synchronization protocols require sensor nodes to store certain number of data in order to achieve high accuracy. Since wireless sensor nodes have strictly constrained size of storage, time synchronization protocols are not suitable for WSN applications, because they demands large amount of storage. Moreover, current time synchronization protocols require sensor nodes to exchange messages with large payload size with each other when synchronizing clocks, which consume both data size and energy. Time synchronization protocols requiring large storage is not preferable to be used on wireless sensor nodes, since they have severely strict storage.

3.4.4 Clock Adjustment

There are a couple of ways to synchronize clocks with or without correcting clocks of sensor nodes as follows: 1) correcting actual clocks, 2) using virtual clocks, and 3) keeping relative time. Problems of each method are stated in the following passages.

- Correcting actual clocks

  Correcting actual clocks of sensor nodes seems to be easy way to synchronize sensor nodes' clocks, however, as Mock et al. claims in their paper [22], it has high overhead. Fig. 3.1 illustrates the problem of correcting actual clocks, where *ta* indicates the time right before synchronization, *tb* and *tc* shows the time of right after clock adjustment, and x indicates scheduled tasks. In case B, some tasks are skipped, while in case C, some tasks are reinvoked due to the clock correction. A lot of applications does not allow skipping or redoing scheduled tasks. Think of a case where tasks are scheduled to invoke every second, and a task is skipped as a result of 0.1 second correction. Next task will be invoked 1.9 seconds after, rather than 1 second after since the last task being invoked, which could mess up a lot of WSN applications' data analysis. In addition, correcting actual clocks require large amount of calculation and energy, thus,

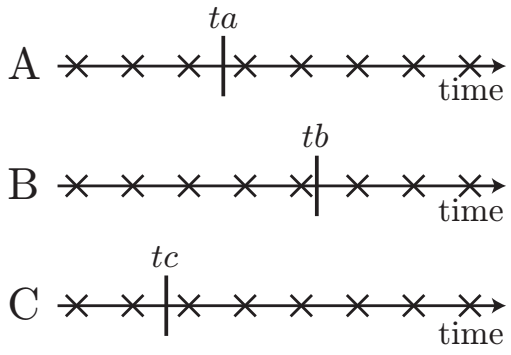
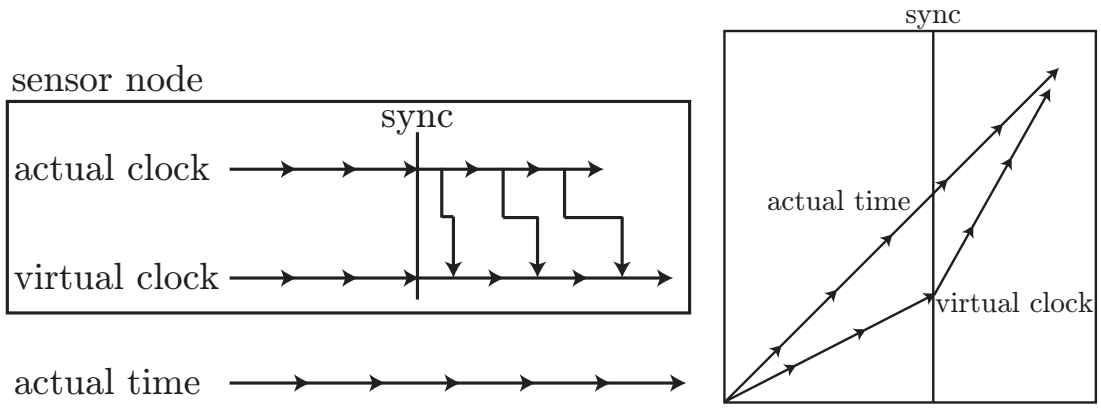


Figure 3.1: Task Problem of Actual Clock Adjustment



(a) Behavior of Virtual Clocks on Sensor Nodes

(b) Behavior of Virtual Clocks After Synchronization

Figure 3.2: Behavior of Virtual Clock

this method is not preferable.

- Using virtual clocks

  Mock et al. proposed the idea of virtual clock in their paper [22]. By using virtual clocks, there would be no need to worry about skipping or redoing scheduled tasks. Fig. 3.2(a) shows how virtual clock operates on sensor nodes. Sensor nodes create a virtual clock, which ticks every time the actual clock ticks, and every events and tasks are invoked according to the virtual clock rather than the actual clock. When time synchronization occurs, sensor nodes investigate if their virtual clocks are ahead, or behind the reference time. If the virtual clocks are behind the reference time, sensor nodes make their virtual clocks go quicker by extending the time of each clock tick, and vice versa. Fig. 3.2(b) shows how virtual clock operates. By storing some synchronization information, sensor nodes can maintain certain accuracy even if they fail to receive some synchronization information. However, the use of virtual clock requires large amount of energy, because sensor nodes need to do calculation every

time the actual clock ticks.

- Keeping relative time

  Keeping relative time on each sensor node is one of the alternatives of clock correction. Many time synchronization protocols [9] [19] use this method to minimize overhead. For instance, sensor nodes running RBS keep the relative time against other sensor nodes in WSNs, and sensor nodes running FTSP keep the relative time against "global time", which is based on the root node's clock. Keeping relative time is said to have low overhead compared to other clock correction methods. However, in order to keep relative time, sensor nodes need to store relative time data, and most protocols require certain number of synchronization information to maintain high accuracy. Minimizing the storage usage is as important as improving the accuracy, thus, suppressing the storage usage should be taken into consideration. For these reasons, the method of keeping relative time is not desirable in perspective of storage usage.

3.5 Summary

This section summarizes this chapter. We explained the target environment which includes two factors; 1) large scale WSNs and 2) existence of sink nodes. We target large scale WSN applications, so that our scheme can be adapted to as many WSN applications as possible. Since most WSN applications have at least one sink node, assuming the existence of sink node is appropriate. WSN application with no sink node is unrealistic, because sensor nodes usually have severely limited size of storage. Then, we describe four requirements of solutions of time precision problem in target environment as follows; 1) low energy consumption, 2) high accuracy, 3) high scalability and 4) small storage usage. Since wireless sensor nodes are usually running with batteries mounted on them, their energy source is severely limited,

and thus, the primary requirement is to lower the energy consumption. Then we brought out the problems of time synchronization in terms of the requirements. We conclude that time synchronization protocols do not fulfill the requirements expect for accuracy, which is their principal goal.

Chapter 4

LECTES: Low Energy Consumption Time Estimation Scheme

4.1 Introduction

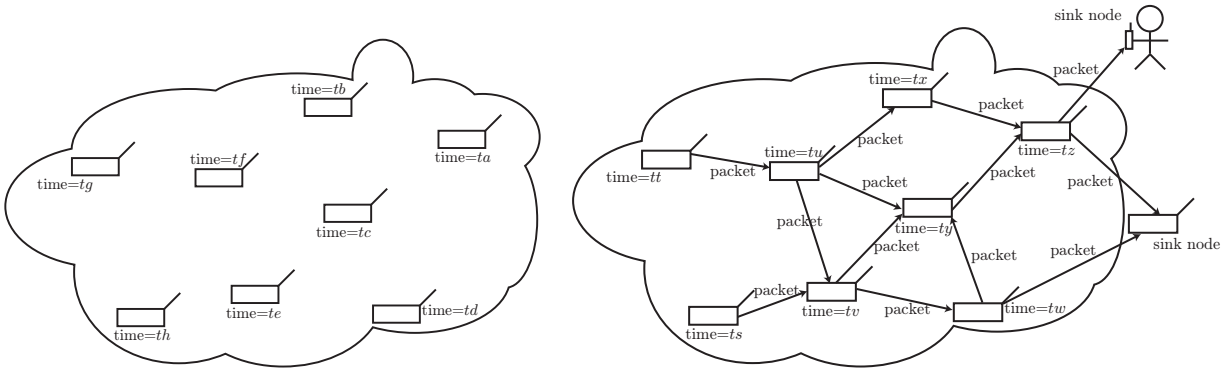
This chapter introduces low energy consumption time estimation scheme, LECTES. First, we explain the time estimation protocol of LECTES. We then compare LECTES against time synchronization protocols in perspective of requirement fulfillment. Finally, we summarize this chapter.

4.2 Time Estimation Protocol of LECTES

This section introduces the time estimation protocol of LECTES. We first mention the overview of LECTES, and then present three phases of it; Delay Set, Delay Add and Time Estimation.

4.2.1 Overview

Fig. 4.1(a) shows a WSN using LECTES. LECTES does not synchronize clocks of sensor nodes in WSNs, hence sensor nodes hold different time from each other. In consequence, LECTES does not provide uniform time for WSNs, or relative time between sensor nodes. Due to this characteristic, LECTES cannot be used in WSN applications where sensor nodes



(a) WSN Using LECTES (b) Packet Transfer in WSN Using LECTES

Figure 4.1: LECTES

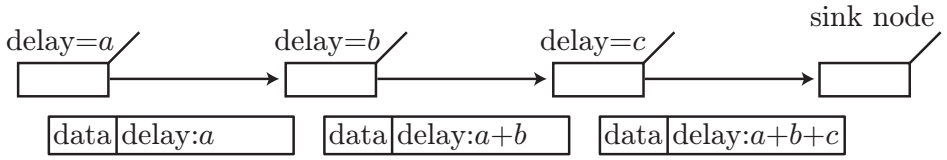


Figure 4.2: Brief Behavior of LECTES

store sensing data on their own storage. However, it should not be a problem, because wireless sensor nodes usually have small storage, and storing data on them is not realistic.

Fig. 4.1(b) illustrates how packets are transmitted to the sink nodes in WSN using LECTES. Sensor nodes running LECTES do not have to include their local time information into the packets, instead, LECTES stores the "delay" information on them. Fig. 4.2 shows the behavior of sensor nodes using LECTES, and how delay information is passed along. We define "delay" as the time required to process data or packets on each sensor node. When a sensor node receives packets, it subtract the delay of packets from the reception time of them in order to estimate the data acquirement time. The estimated time is based on receiver node's clock, not the sender nodes' clocks.

LECTES is designed to solve the time precision problem in WSNs. LECTES assumes applications using large scale WSNs, where each node is equipped with CPU, memory,

sensors, radio chip and batteries. In addition, at least one sink node must be provided in WSN when using LECTES.

4.2.2 Delay Set

Delay Set phase is used by sender nodes. In this phase, when sensor nodes send packets, LECTES estimates the sending time and calculates delay, which is the difference in time between sensing data and sending packets. LECTES stores delay information into the packets, and sends them toward the specified destination. In this phase, users and application providers are in charge of notifying LECTES the data acquirement time. Estimation of sending time must be accurate in order to provide high precision. Therefore, Delay Set phase should take place right before sending packets.

Fig. 4.3 briefly shows how Delay Set phase works. A sender node is depicted at top of the figure. Sender node stores the delay information (a) to the packet, and transmits it.

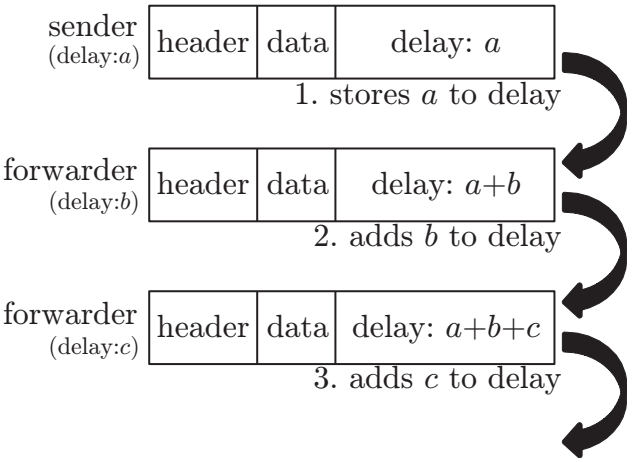


Figure 4.3: Delay Set and Delay Add

4.2.3 Delay Add

Delay Add phase is used by forwarder nodes. In this phase, LECTES estimates the time required to forward a packet. Delay Add is used only when the node supports multi hop, and if the node is not the destination of the packet. LECTES should be doing all the tasks, so that the users and application providers are kept away from cumbersome tasks. If sensor nodes do not support multi hop, creating a multi hop network by hand is a pain in the neck, therefore, eliminating complex work is meaningful to minimize users and application providers tasks.

Fig. 4.3 shortly illustrates how Delay Add phase works. Sender node is depicted at top of the figure, which runs Delay Set; the sender node stores delay (a) in the packet. Then the forwarder node adds the delay (b) to the delay of packet and forwards it, and so on.

4.2.4 Time Estimation

Time Estimation phase is used by receiver nodes. In this phase, LECTES provides the estimated time of data acquirement according to the receiver nodes' clocks. LECTES calculates the estimated time by subtracting the delay of packets from the reception time of them. Note that Time Estimation phase is only used when the destination of the packet is the

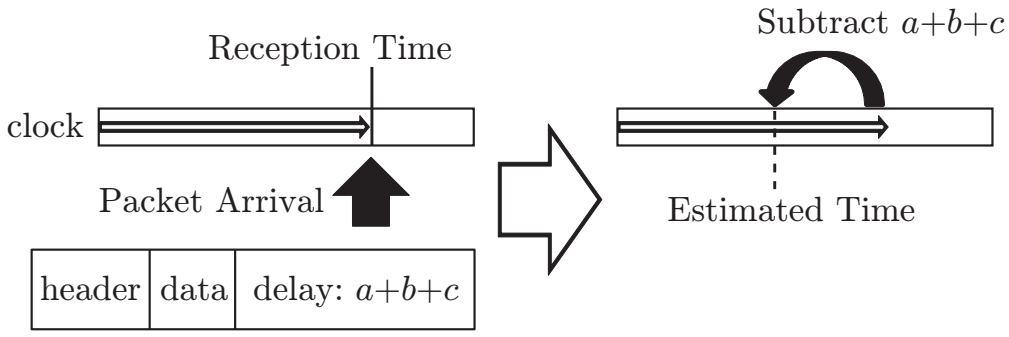


Figure 4.4: Time Estimation

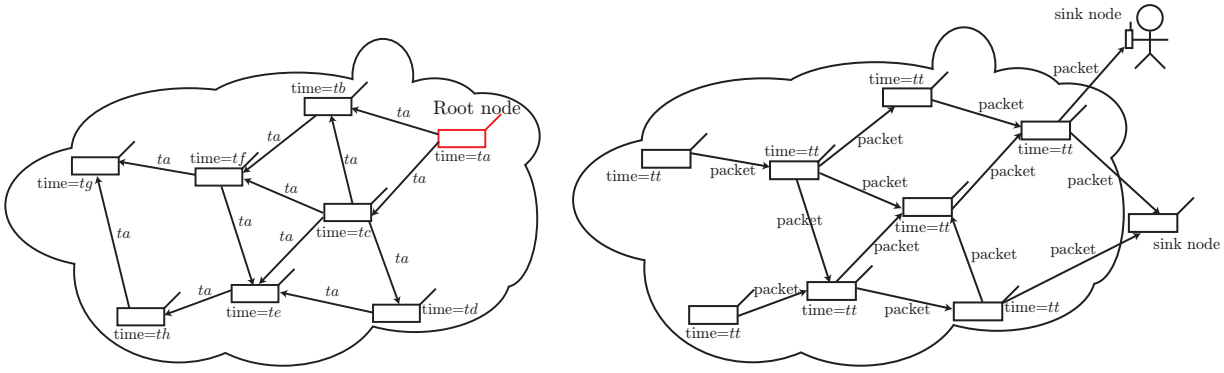
node that received it. LECTES should minimize the effect receiving jitter, so that it can increase the accuracy.

Fig. 4.4 indicates how Time Estimation phase works. When a node receives a packet, it computes the estimated data acquirement time by subtracting the delay of a packet from the reception time of it.

4.3 Comparison of Time Synchronization and LECTES

Related works [9] [10] [22] [19] [24] [30] propose time synchronization protocols which synchronize the clocks of sensor nodes in WSNs. In WSNs, sink node collects sensor data from the sensor nodes using wireless communication and analyzes the obtained data. In the majority of WSN applications, precision of time is extremely important when processing data. Time synchronization is one of the solutions to minimizes the effect of the clock drifting problem in WSNs, although, synchronizing clocks of all sensor nodes in WSNs is waste of resource. We compare LECTES against FTSP, because FTSP is said to be achieving high accuracy with small amount of energy.

Fig. 4.5(a) illustrates how FTSP runs time synchronization. FTSP synchronizes clock of



(a) WSN Using FTSP

(b) Packet Transfer in WSN Using FTSP

Figure 4.5: Behavior of FTSP

root node, located in top right of the figure, to other sensor nodes using flooding model. This figure shows how time of root node (ta) is synchronized to other nodes ideally: in reality, there are some synchronization errors, which cause sensor nodes to have slightly different time from each other. Note that this figure shows only a type of time synchronization protocols. There are many other types of time synchronization protocols, such as a protocol assuming the existence of smart nodes [26] or a protocol requiring large number of message exchanges [9]. Fig. 4.5(b) shows how packets are transferred to the sink nodes. Sensor nodes include their local time information to the packet, and send it toward the sink nodes. By doing so, sink nodes are able to analyze data without being affected by clock drifting problem, because sensor nodes share the same time.

Table 4.1 shows the differences between time synchronization and LECTES in terms of requirements in target environment. Requirements are not fulfilled in the current time synchronization protocols. The primary goal of current time synchronization protocols is to achieve high precision, meanwhile, principal objective of LECTES is to minimize the

Table 4.1: Comparison Between Time Synchronization and LECTES

Requirement	Time Synchronization	LECTES
Energy Consumption	High, due to message exchanges and its frequency (Refer to Sec. 3.4.1)	Extremely low, by eliminating the needs of message exchange
Accuracy	Considerably high, but depends on protocols (Refer to Sec. 2.4)	Considerably High, by removing the error factors as much as possible
Scalability	Low, because of high packet loss rate caused by message exchanges (Refer to Sec. 3.4.2)	High, by removing the needs of message exchange
Storage Usage	Large, for storing multiple data, and possibly information relating to clocks (Refer to Sec. 3.4.3 and Sec. 3.4.4)	Small, because LECTES only requires delay information to be stored

energy consumption. Current time synchronization protocols fail to satisfy requirements other than accuracy; it is known that FTSP achieves the accuracy of 1.48 microseconds and RBS achieves the accuracy of 29.1 microseconds in a single hop network. Energy consumption and scalability is poor, because time synchronization requires to exchange messages among sensor nodes. Moreover, current time synchronization protocols use large amount of storage as a trade off for achieving high accuracy.

LECTES removes the need of exchanging extra messages, which improves the energy consumption and scalability, while minimizing the storage usage. LECTES also achieves high accuracy by minimizing the effects of sending and receiving errors as much as possible. Since sensor nodes using LECTES can use the delay information as a replacement of local time information, the size of packets may be reduced. As shown in the table, LECTES fulfills all the requirements, while time synchronization protocols fails to accomplish them except for the accuracy.

4.4 Summary

In this section, we summarize this chapter. We explain the time estimation protocol of LECTES by showing how it works. LECTES assumes the existence of sink node, which is valid, because majority of WSN applications use sink nodes to collect and analyze data. We then describe three phases of LECTES; 1) Delay Set, 2) Delay Add and 3) Time Estimation. They are used by sender, forwarder and receiver nodes, respectively. We compared time synchronization and LECTES in terms of requirement fulfillment. Time synchronization protocols fail to achieve requirements expect for accuracy, because the primary target of them is to improve the accuracy, and other factors are sacrificed for it. On the other hand, LECTES successfully fulfills all the requirements.

Chapter 5

Design and Implementation of LECTES

5.1 Introduction

This chapter describes the design and implementation of LECTES. First, we discuss where to implement LECTES. Second, the architecture of LECTES is described. Then, we introduce three interfaces provided by LECTES; LectesAMSend, LectesReceive and LectesInfo. Fourth, we explain the available options provided by LECTES. We then explain the usage of LECTES. Finally, this chapter is summarized.

5.2 Where to Implement

This section discusses where to implement LECTES. We first state the platform we are using, and shows the layering of its radio chip. We then explain the method of how LECTES handles the delay information in order to minimize the error.

5.2.1 Platform

Related works propose time synchronization protocols [9] [10] [22] [19] [24] [30], and implement them on mote [7], which is most widely used wireless sensor node. In order to evaluate LECTES against others fairly, we also chose mote to implement LECTES on. There are

Table 5.1: Comparison Between Micaz and Iris

	Micaz	Iris
CPU	Atmega128L	Atmega1281
Flash Memory	128K	128K
Serial Flash	512K	512K
RAM	4K	8K
Radio Chip	CC2420	RF230
Frequency Band	2,400-2,483.5MHz	2,400-2,480MHz
TX Data Rate	250kbps	250kbps

many types of motes, such as Mica2, Micaz, Iris, Telos, etc. Related works use either Mica2 or Micaz, which works on 868/916MHz and 2.4GHz ISM band, respectively. Micaz supports ZigBee, which is a specification for wireless personal area network (WPAN), whose characteristics are low power, low energy and short range. Table 5.1 shows the specification of Micaz and Iris mote. We implement LECTES on Iris mote. As shown in table, there are not much differences between Micaz and Iris mote, except for RAM size and the amount of energy required in sleep mode. We do not use mote in sleep mode, hence, using Iris mote instead of Mica2 or Micaz is reasonable.

Programming motes can be done through TinyOS [27], Xmesh, Contiki, etc. TinyOS is an open source, component-based operating system which is specialized for wireless sensor nodes. Xmesh is provided by Crossbow [8], the company which provides mote. Since TinyOS is more flexible and related work uses it, we implement LECTES on motes using TinyOS. LECTES is programmed with network embedded systems C (nesC), which is an event-driven programming language used in TinyOS.

5.2.2 Layering of Radio Chip

Figure 5.1 illustrates the wiring of RF230 module in TinyOS2.x. For instance, when application uses AMSend interface provided by ActiveMessageLayerC, ActiveMessageLayerC

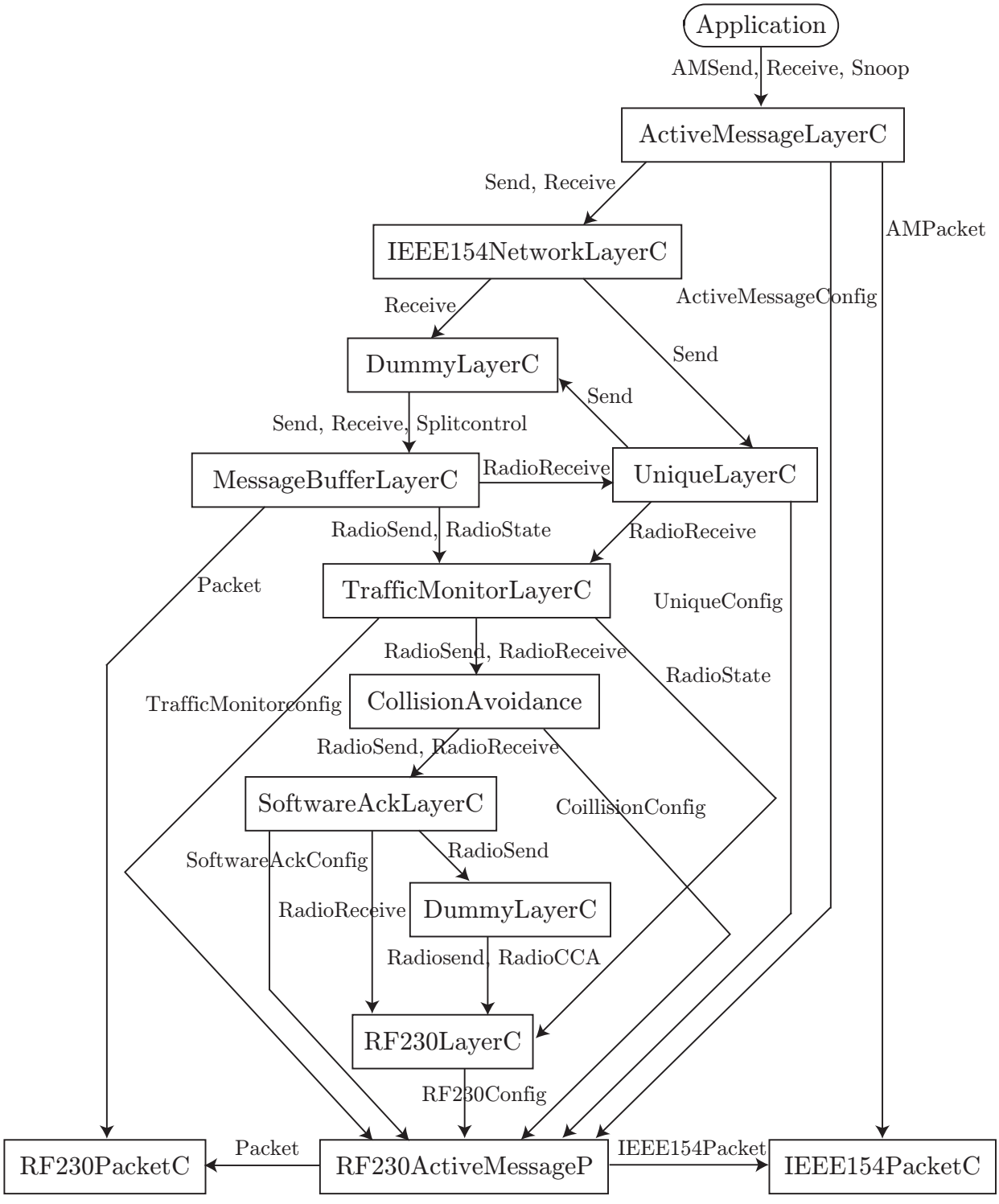


Figure 5.1: Wiring of RF230 Radio Modules

invokes "send" of IEEE154NetworkLayerC, and so on. DummyLayerC in the figure does nothing, but invokes the next module when it is called. Users and application providers may choose to use option LOW_POWER_LISTENING and/or TFRAME_ENABLED, which overrides DummyLayerC. Low power listening (LPL) is provided for those who want to suppress the energy consumed by the radio chip. LPL suppresses the energy usage by turning the radio chip on and off repeatedly. Enabling TFRAME allows sensor nodes to support 6LowPan, which is a IPv6 protocol for low power WPAN. CollisionAvoidance in the figure is either RandomCollisionLayerC or SlottedCollisionLayerC. Default collision avoidance algorithm is RandomCollisionLayerC, which waits for random time when collision occurs. On the other hand, SlottedCollisionLayerC holds time slot which is shared with nearby nodes, and it avoids collision by preempting the time slot. LECTES should not be affected by the use of modules or options, thus, it should be implemented in the lower layer to minimize the error. Therefore, calculation of delay should take place in RF230LayerC.

5.2.3 Delay Management

LECTES uses delay information to estimate the data acquirement time on receiver nodes. We define delay as time taken to process data or packets on each node as follows: for sender nodes, delay is time spent since they sensed data until they send packets containing them, and for forwarder nodes, delay is time spent since they receive packets until they transmit them. In order for receiver nodes to estimate the data acquirement time, delay information must be included in the packets. Since it is required all the time, delay information should be stored in either header or footer of packets. The jitter in time when sending packets is much larger than that when receiving them, thus, we need to eliminate the effect of sending jitter as much as possible. We minimize the error by putting delay information in the footer of the packet, so that the calculation can take place right before sending packets.

Kopetz et al. proposed the error factors of wireless communication [16], and Ganeriwal et al. [10] and Horauer et al. [13] extended it as follows: send time, access time, transmission time, propagation time, reception time and receive time. Send time, access time and transmission time are related to sending packets. Send time is the time spend to request the MAC layer for sending packet, and access time is the time required to access the channel. Transmission time is the time needed to transmit a packet. Propagation time is the time since the packet is transmitted by sending node until it reaches the receiving node. Reception time and receive time are related to receiving packets. Reception time is the time which receiver node requires to receive the packet, and receive time is the time required to notify the application about the reception of packets. Propagation time is negligible, since the speed of radio wave is equivalent to speed of light, and the time of propagation is usually less than 1 microsecond. The time required for receiving packets can be up to few hundred microseconds, while that for sending them can be up to few hundred milliseconds. In order to achieve high accuracy, we store and acquire delay information in MAC layer. By implementing LECTES in MAC layer, it is kept away from the error factors except for the propagation time, which is negligible.

In order to estimate the time accurately, we provide delay information in microseconds precision. Current time synchronization protocols also use microseconds precision clock. They can be used only when CPU is not in sleep mode. LECTES requires clock in microseconds precision only when sensor nodes are sending or forwarding packets, therefore, it does not matter if the microsecond precision clock is available or not when CPU is in sleep mode. On the other hand, when sensor nodes using time synchronization protocols return from the sleep mode, they have totally different time from each other until they get synchronized.

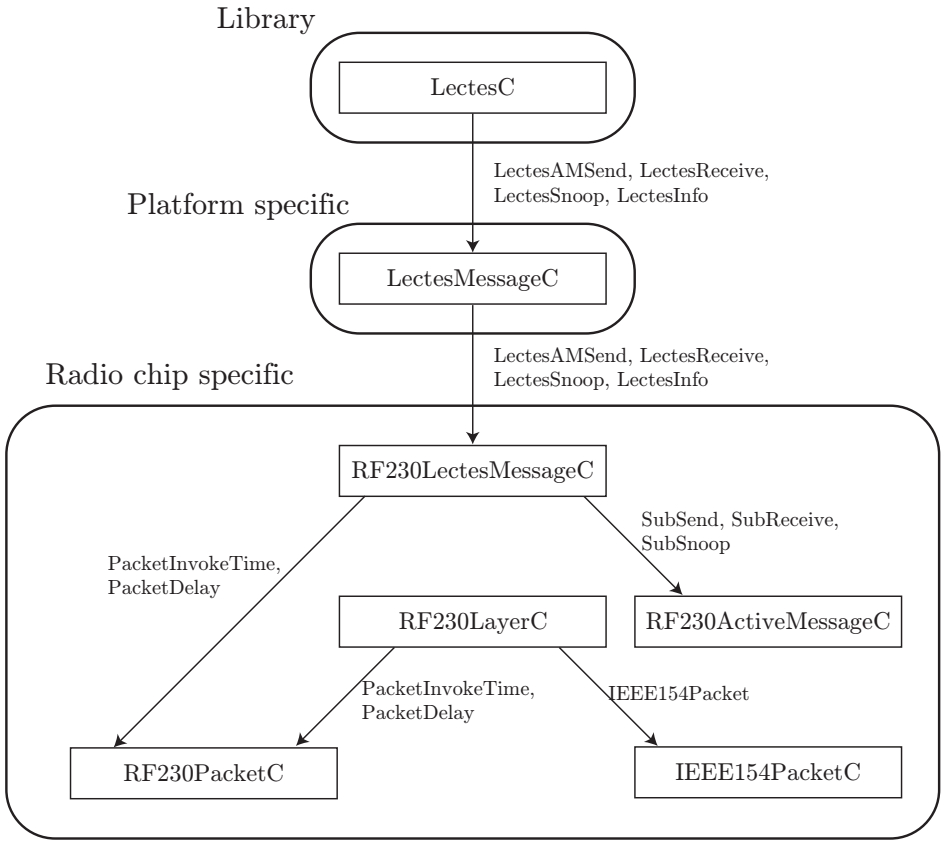


Figure 5.2: Wiring of LECTES

5.3 Architecture of LECTES

Fig. 5.2 illustrates the architecture of LECTES. We implement a TinyOS2.x library called LectesC. LectesC allows users and application providers to use LECTES without complex implementation. LectesC provides users the access to the interfaces required for wireless communication. The interfaces provided by LectesC are connected to LectesMessageC, which is a platform specific component. Currently, LectesMessageC is only available for Iris platform, because we evaluate LECTES on Iris mote. Interfaces of LectesMessageC are connected to a radio chip specific component, RF230LectesMessageC. RF230LectesMessageC is then wired to RF230PacketC and RF230ActiveMessageC, and RF230LayerC is wired to RF230PacketC

and IEEE154PacketC. Note that RF230PacketC, RF230ActiveMessageC, RF230LayerC and IEEE154PacketC are components of RF230 radio chip provided in TinyOS2.x, and We add code to them for LECTES.

Sending and receiving interfaces of LECTES use RF230ActiveMessageC's sending and receiving interfaces (See Fig. 5.1). The information of LECTES can be acquired using LectesInfo interface. Two interfaces, PacketInvokeTime and PacketDelay, are provided by RF230PacketC to control the information of LectesInfo. RF230LayerC acquires the information of LECTES from RF230PacketC, and obtains the information of packets from IEEE154PacketC. LECTES stores delay information into sending packet right before transmitting it in RF230LayerC. LECTES reduces the effect of error by directly accessing to data of packets through IEEE154PacketC.

5.4 Interfaces Provided by LECTES

We implement a TinyOS2.x library for using LECTES. Also, we provide three interfaces for using LECTES as explained below. Interfaces declare functions with commands and events, where commands can be called to invoke specific behavior, and events can be detected by being signalled.

5.4.1 LectesAMSend

LectesAMSend interface is provided for users and application providers to send packets using LECTES. Fig. 5.3 shows the LectesAMSend interface. This interface is equivalent to "AMSend" interface which is offered for ordinary packet sending with grouping ability in TinyOS2.x. Grouping allows multiple WSNs existence in the same place by allowing communication only among the sensor nodes that are in the same group. LectesAMSend requires "lectes_send_time_size" as an argument, which stands for the size of local time used

```
#include <message.h>
#include <AM.h>

interface LectesAMSend <lectes_send_time_size> {

  command error_t send(am_addr_t addr, message_t *msg, uint8_t len,
      lectes_send_time_size invokeTime);

  command error_t cancel(message_t *msg);

  event void sendDone(message_t *msg, error_t error);

  command uint8_t maxPayloadLength();

  command void *getPayload(message_t *msg, uint8_t len);
}
```

Figure 5.3: LectesAMSend Interface

in the application. In TinyOS2.x, time is only provided in unsigned 32 bit format, and it should be provided in microseconds precision in order to achieve high accuracy. Unsigned 32 bit time in micro precision can only count up to about 4,295 seconds, which is quite small for the use in WSN applications. Everything but "send" command is exactly the same as usual sending interface provided in TinyOS2.x; cancel invalidates send, sendDone is signalled when sending is over and maxPayloadLength and getPayload is used to acquire payload information. The send command requires an extra argument, invokeTime, compared to that of ordinary send command. The invokeTime has a format of lectes_send_time_size, and it is set when send command is called. When sensor nodes send a packet using LectesAMSend, they have to pass the data acquirement time as invokeTime. LECTES uses invokeTime to measure the delay, which is calculated by subtracting invokeTime from the transmission time of the packet.

```
#include <TinyError.h>
#include <message.h>

interface LectesReceive <lectes_receive_time_size> {

  event message_t *receive(message_t *msg, void *payload, uint8_t len,
      lectes_receive_time_size estTime);

}
```

Figure 5.4: LectesReceive Interface

5.4.2 LectesReceive

LectesReceive interface is supplied for receiver nodes to receive packets using LECTES. Fig. 5.4 indicates the interface of LectesReceive. This interface is similar to "Receive" interface, which is provided for ordinary packet receiving. LectesReceive needs an argument "lectes_receive_time_size", which should be the format of local time used in the application. LectesReceive only provides receive event, which is signalled when the sensor nodes receive packets. The receive event of LectesReceive has an additional argument, estTime, compared to that of ordinal receive event of Receive interface. The estTime argument provides the estimated time of data acquirement. Note that the estimated time calculated by LECTES is based on the receiver node's clock.

5.4.3 LectesInfo

LectesInfo interface is provided for controlling and acquiring the information of LECTES. Fig. 5.5 illustrates the interface of LectesInfo. LectesInfo requires two arguments; "delay_size" and "time_size", where delay_size should be size of delay, and time_size should be in the format of local time used in the application. LectesInfo is used to control two parameters; invokeTime and Delay. The invokeTime is used when sending and receiving

```
interface LectesInfo <delay_size, time_size> {
  command bool isSet(message_t *msg);
  command void clear(message_t *msg);
  command time_size getInvokeTime(message_t *msg);
  command void setInvokeTime(message_t *msg, time_size time);
  command time_size getEstimatedTime(message_t *msg);
  command delay_size getDelay(message_t *msg);
  command void setDelay(message_t *msg, delay_size delay);
}
```

Figure 5.5: LectesInfo Interface

packets. When sending packets, sender nodes set data acquirement time as invokeTime, and LECTES uses that information to calculate delay. Commands isSet and clear are provided for controlling the flag of invokeTime. Commands setDelay and getDelay are used to control the delay information. Users and application providers can forcefully set delay information by using setDelay command, or acquire the delay information of a packet by using getDelay command. The getEstimatedTime command returns the estimated data acquirement time. Note that delay_size is used for the arguments and return values of commands relating to delay information, and time_size is used for arguments and return values of other commands.

5.5 Available Options

A couple of options are provided by the library of LECTES. These options should be passed to the precompiler by using PFLAGS when compiling a program. The available options are, 1) TIME_W64, 2) LECTES_16BIT_DELAY and 3) LECTES_DEBUG. The description and usage of each are as follows.

- TIME_W64

  This option is provided for those who want to handle time in unsigned 64 bit format. Users may pass TIME_W64 to the precompiler, which would use time in unsigned 64 bit format instead of unsigned 32 bit in the radio modules. Default format of time is only given in unsigned 32 bit. In addition, because LECTES needs data acquirement time in microseconds precision, microseconds precision time in unsigned 32 bit can only count up to about 4,295 seconds, which is not enough for the use in real-world WSN applications. Thus, TIME_W64 option is provided to use unsigned 64 bit time, which allows users and application providers to use LECTES for long time. Since LocalTime interface provided in TinyOS does not support 64 bit local time, we implemented LocalTime64 which returns local time in unsigned 64 bit format. LocalTime64 interface is used in radio modules when TIME_W64 option is used, and users may choose to use this interface. There would be an inconsistency in the format of time, therefore, users or application providers are recommended to use TIME_W64 option and LocalTime64 interface together.

- LECTES_16BIT_DELAY

  This option is provided for those who want to use delay information in unsigned 16 bit format. The default format of delay information is in unsigned 32 bit. By using this option, users of LECTES can suppress the energy used in packet transmission. Panthanchai et al. measured the amount of energy consumed by transmitting packets with different payload size on Telos mote [23]. Although their experiment used different type of mote, the trait of energy consumption is similar to Iris mote, because they are also using TinyOS. They claim that the amount of energy consumed is proportional to the size of packet's payload. Even though the difference in size is only 16 bit, it can

have a huge effect on saving energy when transmitting thousands of packets. Unsigned 16 bit integer in microsecond precision can only count up to about 65 milliseconds, however, it should be long enough for delay in a lot of small scale or single hop WSN applications. Thus, the use of this option is strongly recommended in order to reduce the energy consumption, especially in applications using small sized WSNs.

- LECTES_DEBUG

  As the name of this option indicates, this option is provided for debugging LECTES. By default, LECTES deletes the delay information after successfully sending the packets. This operation avoids sensor nodes to mistakenly applying Delay Add instead of Delay Set. By using this option, LECTES leave the delay information as is, which enables users to acquire delay information. Users of LECTES must delete the delay information manually by using setDelay command provided in LectesInfo interface, unless, the delay information most likely be accumulated. This option is for debugging LECTES, and normally, users of LECTES do not have to worry about using this option.

5.6 Usage of LECTES

This section describes the behavior of LECTES on the sensor nodes. We explain the usage of LECTES in detail for sender, forwarder and receiver nodes.

5.6.1 Sender Node

For sender nodes, LECTES runs Delay Set (See Sec. 4.2.2). When sender nodes use LECTES, they need to notify LECTES the time when data are acquired, which is done through the use of send command of LectesAMSend interface (Fig. 5.3). When sensor nodes send packets, LECTES estimates the transmission time in local clock, and store the delay information, which is calculated by subtracting invokeTime from the estimated transmission time.

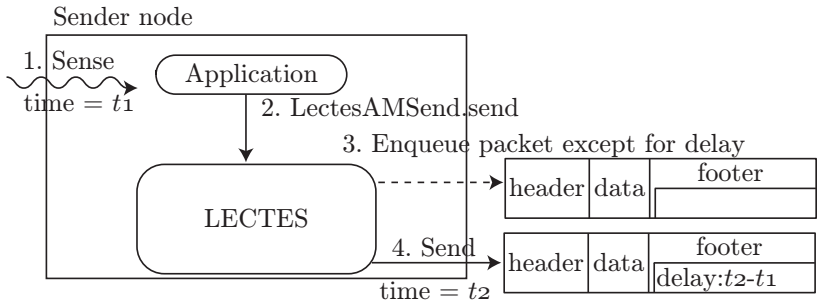


Figure 5.6: Sender Node Using LECTES

Fig. 5.6 indicates how LECTES works on sender nodes. Behavior of sender nodes and LECTES is explained in the following:

1. Sense
   When sensor nodes obtain data by using sensors attached to them, they store the data acquirement time ($t1$).
2. LectesAMSend.send
   Sender nodes notifies LECTES the data acquirement time ($t1$). This is done by using send command of LectesAMSend interface.
3. Enqueue packet except for delay
   Sender nodes enqueue packet's header and data, leaving the delay information.
4. Send
   LECTES acquires the current time ($t2$) and calculates the delay ($t2 - t1$). Then it enqueues the delay information and transmit the packet.

5.6.2 Forwarder Node

For forwarder nodes, LECTES runs Delay Add (See Sec. 4.2.3). Users and application providers do not have to do anything for forwarder nodes, because it is taken care of by LECTES. When sensor node receives a packet, LECTES acquires its reception time and stores it in invokeTime field. If the sensor node which received the packet is not the destination of it, that node forwards it to the destination. When the node is forwarding the packet,

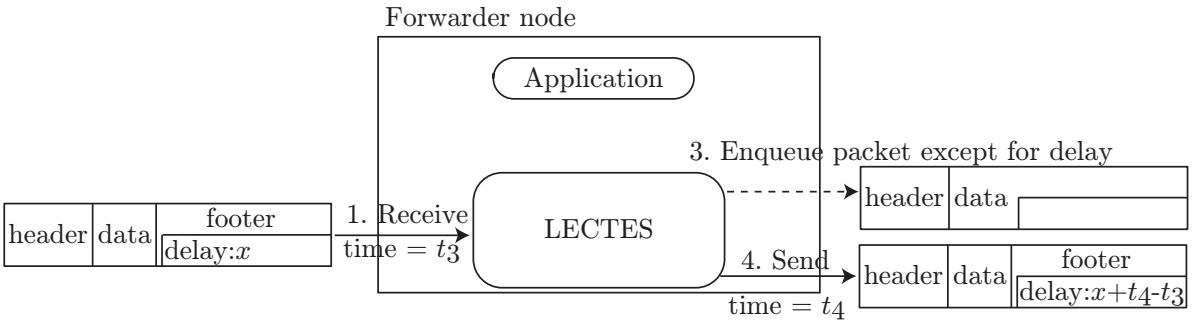


Figure 5.7: Forwarder Node Using LECTES

LECTES works similarly to sender nodes. When transmitting packet, LECTES adds delay, instead of storing it.

Fig. 5.7 shows the role of LECTES for forwarder nodes, and its flow is described below:

1. Receive
   When Sensor node receives a packet, LECTES obtains its reception time ($t3$).
2. Enqueue packet except for delay
   Sender nodes enqueue packet's header and data, leaving the delay information.
3. Send
   LECTES acquires the current time ($t4$) and adds it to the delay ($x + (t4 - t3)$). Then it enqueues the delay information and transmit the packet.

5.6.3 Receiver Node

For receiver nodes, LECTES runs Time Estimation (See Sec. 4.2.4). When receiver nodes receive packets from sensor nodes, LECTES obtains their reception time. LECTES then subtracts delay stored in packets from the reception time to obtain the estimated data acquirement time. The estimated time of packets is based on the receiver nodes' clock. The estimated time is notified to the receiver nodes by signalling the receive event of LectesReceive interface.

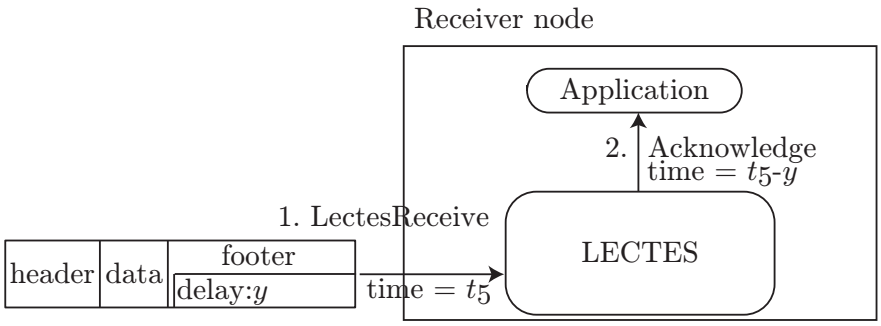


Figure 5.8: Receiver Node Using LECTES

Fig. 5.8 depicts how LECTES works for receiver nodes. The role of LECTES is described in the following passages:

1. Receive
   When Sensor node receives a packet, LECTES obtains its reception time ($t5$). LECTES calculates the estimated time according to receiver nodes' clock, by subtracting the delay of a packet from the reception time of it.
2. Acknowledge
   LECTES acknowledges the estimated time ($t5 - y$) by signalling the receive event of LectesReceive interface.

5.7 Summary

This section summarizes this chapter. First, we state that we should implement LECTES on Iris mote, using TinyOS. Although mote is most widely used wireless sensor node, there are many types of it, such as Mica2, Micaz, Iris and Telos mote. We conclude that implementing LECTES on Iris mote is reasonable, because the performance of them are similar to each other. Using TinyOS is proper, because it allows flexible implementation, and current time synchronization protocols also use it. Then, the layering of Iris mote's radio chip, RF230, is illustrated and discussed. We conclude that the delay information should be calculated

and stored in the footer of packets, so that we can minimize the effects of error relating to sending and receiving packets. We implement a TinyOS library for using LECTES, and its architecture is illustrated. LECTES provides three interfaces, LectesAMSend, LectesReceive and LectesInfo, and each of them are described in detail. The available options offered by LECTES are brought up, and the usage of each is explained. Finally, we show the usage of LECTES for sender, forwarder and receiver nodes.

Chapter 6

Evaluation of LECTES

6.1 Introduction

In this chapter, we present the evaluation of LECTES. First, we explain the purpose of the evaluation. Second, methodology of evaluation, comparison targets and evaluation items are explained in detail. Then, we show the result of the evaluation in terms of evaluation items for each comparison targets. Finally, we summarize this chapter.

6.2 Purpose of the Evaluation

In this Section, we introduce the purpose of the evaluation. The ultimate goal of LECTES is to solve the time precision problem in WSNs. Therefore, we need to check the validity of LECTES as a solution for time precision problem. The adequacy of a solution is proved by measuring the accuracy of it. Required precision differs for each WSN application, however, achieving high accuracy allows LECTES to be used in greater number of them. Therefore, we evaluate the achieved precision of LECTES and time synchronization protocol, and compare against each other.

The primary goal of LECTES is to minimize the energy usage. Thus, through the evaluation, we measure the energy consumption of LECTES and compare it against that of time synchronization protocol. Comparing the energy consumption would prove how much energy

can be saved by using LECTES instead of using time synchronization protocols.

In order for LECTES to be used as a solution for time precision problem, it has to fulfill the requirements other than the energy usage and accuracy as well. We evaluate LECTES in perspective of energy consumption, accuracy, scalability and storage usage, because, all of these factors affects the validity of a scheme as a solution for time precision problem.

6.3 Evaluation Methodology

This section describes the methodology of our evaluation. The overview of methodology is described, followed by the comparison targets and evaluation items.

6.3.1 Evaluation Environment

This section shows the evaluation environment and the basic methodology of evaluation. Table 6.1 shows the evaluation environment. We used the most updated versions of nesC, TinyOS and AVR-GCC as of Aug. 2009. We implemented LECTES on Iris mote [6], a successor of Micaz mote, which has Atmega1281 as a CPU and RF230 as a radio chip.

In our evaluation, we use twelve Iris mote as sending nodes, and one Iris mote as a sink node. The sink node is connected to a personal computer with MIB520. MIB520 is a USB gateway that provides personal computers the connectivity to Iris, Micaz and Mica2 mote.

Table 6.1: Evaluation Environment

Operating Systems	Windows XP SP2
nesC Version	1.3.0
TinyOS Version	2.1.0
AVR-GCC Version	4.1.2.1
Platform	Iris mote (XM2110J)
CPU	Atmega1281
Radio Chip	RF230

Sending nodes are attached with MTS300, a sensor board provided by Crossbow [8], which can acquire illuminance, temperature, acceleration, etc. We want LECTES to be used in a real world WSN application, hence, we made sender nodes to obtain the illuminance value and include it in the packets that are being transmitted. We set the frequency of sensing and packet transmission to 1Hz.

Packets transmitted by sender nodes are collected by the sink node, which is connected to a personal computer. Those packets are not processed by other sender nodes, because the destination of the packets is specified when transmitting them. Sink node is installed with an application that stores certain number of packets transmitted by the sender nodes and sends them sequentially to the connected personal computer. The personal computer stores received information on it whenever it receives data from the sink node. Since all data are saved on the personal computer, we evaluate LECTES and other targets by using the data stored on it. Because we have to evaluate the accuracy of FTSP and LECTES, we do not have a uniform experimental methodology, however, each comparison target is ensured to be evaluated fairly.

6.3.2 Comparison Targets

This section explains three comparison targets of the evaluation, a simple, FTSP and LECTES application. The evaluation procedure is explained for each comparison target with pseudo code.

Simple Application

First comparison target is a simple WSN application. In this application, sender nodes acquire illuminance value from the sensor board attached to them at 1Hz. Whenever they obtain illuminance value, they send a packet containing it to the sink node with their ID,

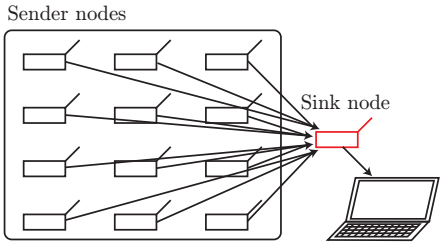


Figure 6.1: Evaluation Methodology of Simple Application

```
...
implementation {
    ...
    event void Boot.booted () {
        call Timer.periodic(1Hz);
    }

    event void Timer.fired (error_t err) {
        packet->nodeid = TOS_NODE_ID;
        packet->illuminance = call sensor.get();
        packet->voltage = call voltage.get();
        packet->local_time = call localtime.get();
        packet->seq = ++seq;

        call AMSend.send(SINK_NODE, packet, sizeof(PACKET));
    }
}
```

Figure 6.2: Pseudo Code of Sender Nodes in Simple Application

time, sequence number and voltage of the batteries.

Fig. 6.1 shows how sink node collects data in the simple application. Each sensor node sends packets at 1Hz to the sink node. Those packets are stored on the personal computer through the sink node. This methodology ensures the frequency of sensing and packet transmission to be 1Hz. Note that the invoke node is connected to MIB520, so that it is supplied with semipermanent energy source.

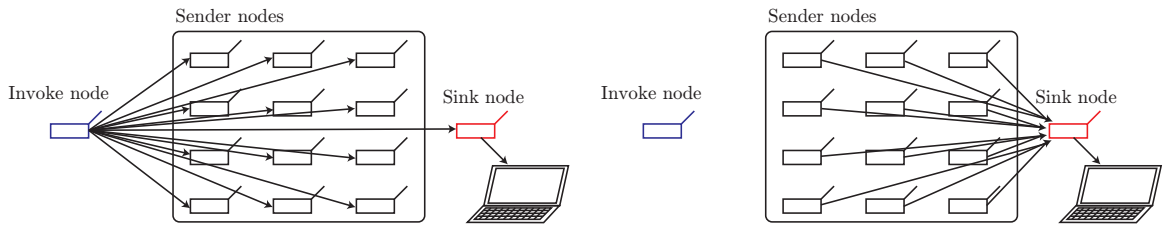
Fig. 6.2 shows the pseudo code of sender nodes in the simple application. The timer is set to invoke at 1Hz when they boot up. Whenever the timer fires, sensor nodes store information into a packet and sends it to the sink node. Note that some events and commands are

abbreviated in the figure.

FTSP Application

Second comparison target is a WSN application using FTSP. FTSP is said to be achieving high accuracy with considerably small amount of energy. Therefore, we chose FTSP as a time synchronization protocol to be compared against LECTES. This application mimics simple application's behavior, except, it runs time synchronization at specific frequency. We set the frequency of time synchronization to 0.8Hz, which is the minimum frequency to ensure 100 microseconds precision, assuming the maximum possible drifting in motes' clocks.

Since we want to evaluate the accuracy of FTSP and LECTES, we have to change the way we collect data from how it is done in the simple application. Procedure of gathering data in the simple application can be applied to FTSP and LECTES application as well, although, with that methodology, we cannot evaluate the achieved accuracy of them. In order to solve this problem, we provide one sensor node as an invoke node. Invoke node works exactly the same manner as the sender nodes in simple application, except, invoke node broadcasts packets rather than sending them to the sink node. When sender nodes receive packets from the invoke node, they modify the packets according to their own information (ID, illuminance value and voltage of batteries), and transmit them to the sink node. Sender nodes store the reception time of packets into the forwarding packets, which is used for accuracy evaluation.



(a) Invoke node sending packets (b) Sender nodes forwarding packets

Figure 6.3: Evaluation Methodology of FTSP and LECTES Application

The reception time should be the same for the packet that has the same sequence number, because the propagation time is negligible and the sender nodes receive the same packet from each other. In order to evaluate the energy consumption of FTSP application fairly, when sender nodes find out that they did not receive some packets from the invoke node, they send the missed number packets to the sink node. This methodology guarantees sender nodes to sense and send packets at 1Hz, while allowing the evaluation of FTSP's accuracy.

Fig. 6.3(a) illustrates how invoke node broadcasts packets, and Fig. 6.3(b) shows how sender nodes forward the received packets to the sink node. Invoke node transmits packets at 1Hz, hence, the sender nodes are set to sense and send packets at 1Hz as well. The sink node ignores the packet transmitted by the invoke node. Note that the invoke node is connected to MIB520, so that it is supplied with semipermanent energy source.

Fig. 6.4 indicates the pseudo code of invoke node and sender nodes in FTSP application. We define the frequency of time synchronization to be 0.8Hz by using PFLAGS in the Makefile, which is passed to the pre-compiler. The parameters of FTSP other than the frequency of time synchronization are left as default. When the sensor nodes are booted, TimeSyncC is invoked, which is the time synchronization component provided in the FTSP library. As expressed in the figure, FTSP can be used appreciably simply.

LECTES Application

The last comparison target is a WSN application running LECTES. The base of this application is equivalent to the simple application, however, this application runs LECTES on top of it. The evaluation methodology is the same as the FTSP application. This method is used in order to evaluate the accuracy of LECTES while having the LECTES application to be fairly evaluated against others. Sender nodes in LECTES application do not include the reception time in the packets, instead, they set the reception time as the invokeTime.

By doing so, the accuracy of FTSP and LECTES application are evaluated evenly, because they both use the reception time of packets as the reference of accuracy evaluation. Fig. 6.3 shows the evaluation setup of LECTES. Sensor nodes in LECTES is set up exactly the same as those in FTSP.

Fig. 6.5 illustrates the pseudo code of invoke node and sender nodes in LECTES appli-

```
----------Makefile----------
PFLAGS += FTSP_FREQUENCY 0.8Hz
----------configuration----------
...
implementation {
    ...
    components MainC, TimeSyncC;
    FTSPApplication.Boot->MainC;
    MainC.SoftwareInit->TimeSyncC;
    TimeSyncC.Boot->MainC;
}
----------module----------
...
implementation {
    ...
    event void Boot.booted () {
        if (TOS_NODE_ID == INVOKE_NODE_ID)
            call Timer.periodic(1Hz);
    }

    event void Timer.fired (error_t err) {
        packet->nodeid = TOS_NODE_ID;
        packet->seq = ++seq;

        call AMSend.send(BROADCAST, packet, sizeof(PACKET));
    }

    event message_t *Receive (message_t *msg, void *payload, uint8_t len) {
        packet->nodeid = TOS_NODE_ID;
        packet->illuminance = call sensor.get();
        packet->voltage = call voltage.get();
        packet->time = timestamp(reception time);
        packet->seq = recvpacket.seq;

        call AMSend.send(SINK_NODE, packet, sizeof(PACKET));

        if ((seq_diff = packet->seq - prev_seq) > 1)
            while (--seq_diff)
                call AMSend.send(SINK_NODE, packet, sizeof(PACKET));
        prev_seq = packet->seq;
    }
}
```

Figure 6.4: Pseudo Code of Invoke Node and Sender Nodes in FTSP Application

```
...
implementation {
    ...
    event void Boot.booted () {
        if (TOS_NODE_ID == INVOKE_NODE_ID)
            call Timer.periodic(1Hz);
    }

    event void Timer.fired (error_t err) {
        packet->nodeid = TOS_NODE_ID;
        packet->seq = ++seq;

        call LectesAMSend.send(BROADCAST, packet, sizeof(PACKET), time);
    }

    event message_t *LectesReceive (message_t *msg, void *payload, uint8_t len,
        lectes_receive_time_size_t estTime) {
        packet->nodeid = TOS_NODE_ID;
        packet->illuminance = call sensor.get();
        packet->voltage = call voltage.get();

        call LectesAMSend.send(SINK_NODE, packet, sizeof(PACKET),
            timestamp(reception time));

        if ((seq_diff = packet->seq - prev_seq) > 1)
            while (--seq_diff)
                call LectesAMSend.send(SINK_NODE, packet, sizeof(PACKET),
                    timestamp(reception time));
        prev_seq = packet->seq;
    }
}
```

Figure 6.5: Pseudo Code of Invoke Node and Sender Nodes in LECTES Application

cation. If a sensor node is set as the invoke node, it works in the same exact manner as the invoke node in FTSP application. When the sender nodes receive a packet from the invoke node, they modify the packet according to their own information, and forward it to the sink node. Then, they check if they successfully received previous packets transmitted by the invoke node, and if they did not receive some of them, they send the missed number of packets to the sink node.

6.3.3 Evaluation Items

In this section, we explain the evaluation items. We propose energy consumption, accuracy, scalability and storage usage as evaluation items. Each of them are described with the reason

we chose it and its evaluation method.

- Energy Consumption

  The primary goal of LECTES is to reduce the energy consumption, thus, we compare the energy usage of LECTES application against that of the simple and FTSP application. Since the most notable characteristic of wireless sensor nodes is having severely limited energy source, the energy consumption should be considered with care. One way to measure the energy consumptions is to run the applications for certain amount of time and to calculate the differences in batteries' voltage. However, the voltage of batteries acquired in TinyOS application has large jitter, which makes it unreliable for evaluating the energy consumption. Therefore, We evaluate the energy consumption of each application by measuring sensor nodes' lifetime. We used fully charged rechargeable batteries to evaluate the comparison target fairly.

  Since the clocks of sensor nodes tick at different rate, we made the sink node to store its local time whenever it receives packets. Lifetime is calculated by subtracting the arrival time of the first packet from that of the last packet. A sensor node may be alive even if it is unable to transmit packets, however, sensor node with no communication ability is equivalent to a dead node in WSN applications. Therefore, calculating the lifetime of sensor nodes using the arrival time of the final packet is valid.

- Accuracy

  Achieved accuracy is as important as the energy consumption. Current time synchronization protocols aim to reduce the jitter between the sensor nodes' clocks in WSNs. On the other hand, the goal of LECTES is to accurately estimate the data acquirement time. In FTSP application, we evaluate the accuracy by measuring the differences in the reception time of packets on sender nodes. In LECTES application, we evaluate

the accuracy by measuring the differences in the estimated time of packets that have the same sequence number on the sink node. We ensure that LECTES and FTSP are evaluated equally, by using the reception time of packets as references to evaluate the accuracy in both of them.

- Scalability

  Since our target environment is large scale WSNs, high scalability should be guaranteed. By supporting large scale WSN applications, WSN applications with various scale can also be supported. In order for WSN applications to have high scalability, accuracy in a multi hop network should be high and the chance of packet collision should be minimized. The packet loss rate shows the quality of the wireless network, thus, it is meaningful to minimize it especially in multi hop networks, because greater number of packets are being exchanged in them. We evaluate the scalability by measuring the accuracy in a multi hop network and the packet loss rate in a single hop network.

- Storage Usage

  Sensor nodes are built to be as small as possible, although, the size of their storage is severely restricted as a trade off. Because the size of storage is limited, requiring large amount of storage is not desirable. We evaluate the storage usage by measuring the size of the compiled program for Iris mote. In addition, the size of packets required in each application is compared.

6.4 Evaluation Results

In this section, we present the results of the evaluation. We first explain the evaluation results briefly. Then, the evaluation results of four evaluation items are particularly explained and discussed.

Table 6.2: Concise Result of Evaluation

		Simple	FTSP	LECTES
Energy Consumption	Sum of Lifetimes	100.0%	85.1%	101.9%
Accuracy	Average Error	n/a	1.618 μs	0.488 μs
	Maximum Error	n/a	13 μs	11 μs
	Standard Deviation	n/a	1.195	0.744
Scalability	Average Error (9 hop)	n/a	7.174 μs	2.538 μs
	Maximum Error (9 hop)	n/a	49 μs	21 μs
	Standard Deviation (9 hop)	n/a	2.052	1.173
	Packet Loss Rate	1.509%	2.851%	1.510%
Storage Usage	ROM	100.0%	155.4%	99.0%
	RAM	100.0%	187.2%	103.5%
	Packet Size	100.0%	154.7%	79.0%

6.4.1 Overview of the Evaluation Results

Table 6.2 shows the concise evaluation results for each evaluation item. Energy consumption is evaluated by measuring the sensor nodes' lifetime. In the table, we express the sum of sensor nodes lifetime in simple application as 100%, and express that of FTSP and LECTES application in comparison to that. Therefore, the larger the sum of sensor nodes' lifetime, the lower the energy consumption is. LECTES successfully decreased the energy consumption compared to FTSP by about 17%, which shows the superiority of LECTES in perspective of energy consumption. Accuracy is evaluated by measuring the average, maximum and standard deviation of errors in a single hop network. The error of accuracy is not evaluated in the simple application. It is better to have smaller values for the error factors, because, they express how errors are determined. As shown in the table, LECTES successfully decreased the average, maximum and standard deviation of errors, which makes LECTES superior to FTSP in terms of accuracy in a single hop network. The scalability is evaluated by the error factors in a 9 hop network and the packet loss rate in a single hop network. The error factors in multi hop network show that LECTES is superior to FTSP, however, we still have

to improve the LECTES's accuracy in multi hop network so that it can be adapted to greater number of WSN applications. The packet loss rate is examined against that of the simple application, because the simple application is the base of FTSP and LECTES application. Note that the lower the packet loss rate, the better the scheme is, because it indicates the frailty of the WSNs. The result of packet loss rate expresses that LECTES is better than FTSP, because its packet loss rate is smaller than that of FTSP. The size of ROM, RAM and packets are measured to evaluate the storage usage. It is better for sensor nodes if the scheme requires small amount of storage. FTSP requires peculiarly large amount of storage, and it has to exchange messages other than data messages, which makes FTSP undesirable. On the other hand, ROM and RAM required by LECTES is equivalent to simple application, and LECTES successfully reduces the packet size, which makes it superior to FTSP in terms of storage usage. The detailed results of evaluation are stated in the following passages.

6.4.2 Energy Consumption

We evaluate the energy consumption by measuring the lifetime of sensor nodes. Since the tasks of simple, FTSP and LECTES application is the same, the lifetime of sensor nodes reflects the energy consumption of each. Lifetime is measured at the sink node, by subtracting the reception time of the first packet from that of the last packet.

Fig. 6.6 shows the lifetime of sensor nodes in each comparison target, where x-axis indicates the sensor node's ID, and y-axis shows the lifetime of sensor node in second precision. The lifetime of sensor nodes used in the simple application lies between 370,217 and 406,681 seconds, and the average lifetime is 385,988 seconds. The lifetime of sensor nodes differs significantly, and because they are placed in the same environment, the difference in lifetime must be caused by the characteristics of the sensor nodes and/or the batteries. In order to evaluate each target fairly, we used sensor nodes and the batteries as a pair throughout

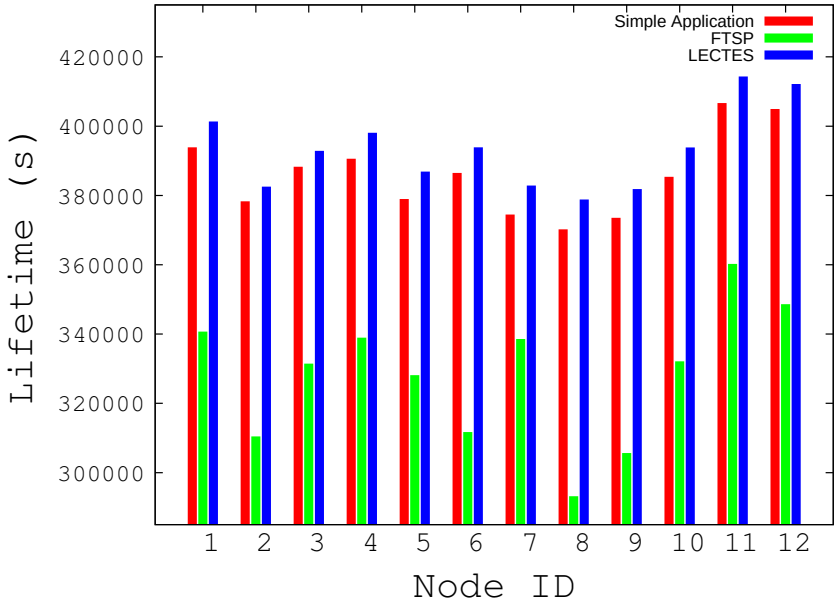


Figure 6.6: Lifetime of Sensor Nodes

Table 6.3: Sum of Sensor Nodes' Lifetime

	Simple Application	FTSP	LECTES
Sum of Lifetime	4,631,864 s	3,939,832 s	4,719,607 s

the evaluation. We used fully charged brand-new rechargeable batteries in the experiment, therefore, sensor nodes are set up identically in each comparison target.

Table 6.3 indicates the sum of sensor nodes' lifetime. The sum of sensor nodes' lifetime in the simple application came out to be 4,631,864 seconds. The sum of sensor nodes' lifetime in FTSP application is 3,939,832 seconds, which is about 85% of that of the simple application. Meanwhile, the sum of sensor nodes' lifetime in LECTES application is 4,719,607 seconds, which is barely longer than that of the simple application. LECTES successfully increased the lifetime of sensor nodes by about 2% compared to the simple application. The reasons of slight increase in sensor nodes' lifetime in LECTES application is discussed in Sec. 6.4.5

In conclusion, LECTES application increased sensor nodes' lifetime by about 2%, while

FTSP application decreased it by about 15% compared to that of simple application. Therefore, we conclude that LECTES is superior to FTSP in terms of energy consumption. Since energy consumption is one of the most important factors of wireless sensor nodes, the fact that LECTES decreases the energy consumption is a huge advantage. Meanwhile, FTSP increased the energy consumption, which is a disadvantage when it is used in WSNs.

6.4.3 Accuracy

Fig. 6.7 shows the time of 1 second in each sensor nodes, where x-axis shows the sensor nodes' ID and y-axis illustrates the time in microseconds. The result is based on the data of the simple application. Time of sensor nodes is calculated by dividing the differences in reception time of first and last packet, by the difference in sequence number of them. Note that the time of sensor nodes indicated in the figure is based on the sink node's clock. Since sender nodes are scheduled to transmit packets at 1Hz, the calculated time is equivalent to 1 second in each sensor node. The gap between the packets receptions has large jitter,

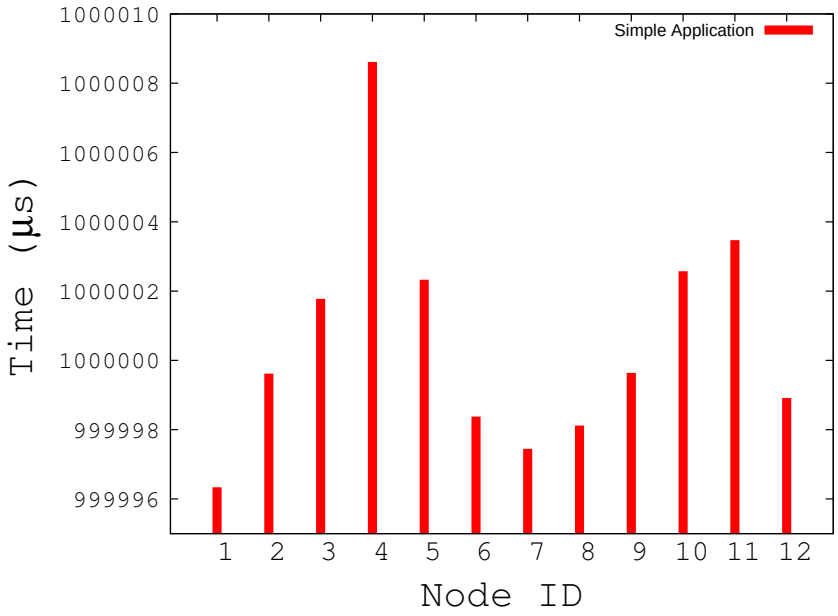


Figure 6.7: Time of 1 Second in Sensor Nodes

Table 6.4: Accuracy Evaluation Results

		FTSP	LECTES
Accuracy	Average Error	1.618 μs	0.488 μs
	Maximum Error	13 μs	11 μs
	Standard Deviation	1.195	0.744

however, it is negligible, because the number of data is large. Differences in time of sensor nodes is the result of clock drifting. For instance, the difference in time of node 1 and 2 is 3.281 microseconds, which means that the time of node 1 and 2 draws apart from each other by 3.281 microseconds per second. The largest drift is observed between node 1 and 4, which was 12.275 microseconds. The maximum drift in sensor nodes' clocks would be larger if greater number of sensor nodes are involved.

Table 6.4 indicates the accuracy of FTSP and LECTES. The average, maximum and standard deviation of errors are compared. Note that the data used for evaluating FTSP's accuracy is "synchronized" data, which is claimed by FTSP. LECTES remarkably decreases the average error; the average error of LECTES is 0.488 μs, and that of FTSP is 1.618 μs. This shows that LECTES is much superior to FTSP in terms of single hop accuracy, because the average error of LECTES is about a third of that of FTSP. In FTSP application, the average accuracy depends on the frequency of time synchronization; the lower the frequency of time synchronization, the lower the accuracy becomes. This occurs because the sensor node's clock drift between the synchronization. On the other hand, LECTES establishes an uniform accuracy under any circumstances, because it does not involve synchronization. LECTES sends the delay information in the very end of transmission to reduce the effect of error when sending packets, thus, it is understandable that LECTES have higher accuracy than FTSP.

The maximum error of FTSP is 13 μs, while that of LECTES is 11 μs. In some WSN

application, the maximum error is more important than the average error, since it could be a critical issue when processing data. For example, in case of counter sniper application [12], using the data with maximum error can be life threatening. LECTES should decrease the maximum error in order to be more reliable. Therefore, we should improve LECTES implementation in order to minimize the maximum error.

The standard deviation of FTSP and LECTES is 1.195 and 0.744, respectively. It is better if the standard deviation of errors is small, because it indicates the dispersion of the errors. Fig. 6.8 shows distribution of errors in FTSP and LECTES application, where x-axis expresses error in microsecond precision, and y-axis represents the percentage of each error. The figure shows that the majority of the errors in LECTES lies between -2 to 2, while that in FTSP lies between -5 to 5. This indicates that LECTES is better than FTSP, because LECTES has extremely limited number of absolute errors greater than 3, while FTSP has certain number of errors between -5 to 5. In LECTES application, the sum of absolute

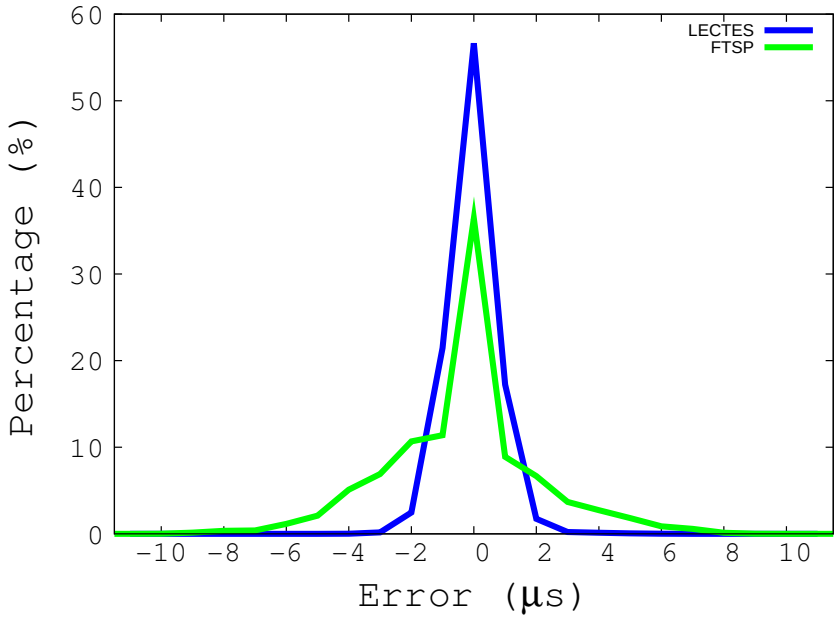


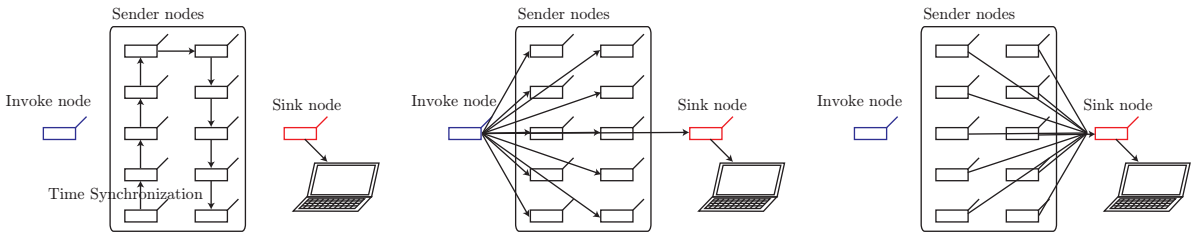
Figure 6.8: Standard Deviation of Errors

errors larger than 2 is only about 0.5%, while that of FTSP is about 16%. This shows that LECTES is much superior to FTSP in terms of error distribution. LECTES only has a few number of large errors, however, we should eliminate those errors to improve LECTES.

To conclude, LECTES decreased the average, maximum and standard deviation of errors in relation to that of FTSP. The fact that LECTES decreased the error factor proves that LECTES is more desirable than FTSP in perspective of accuracy.

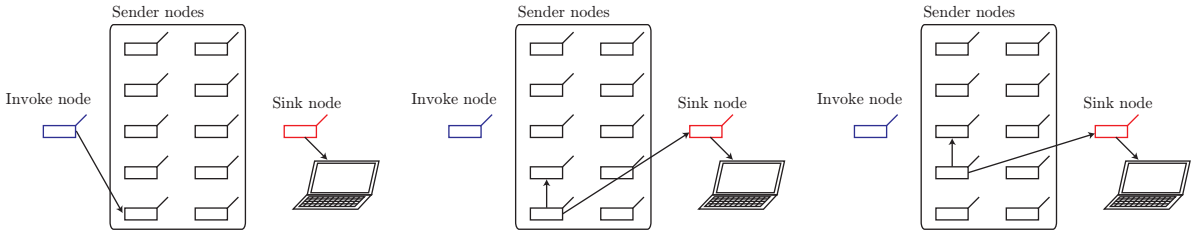
6.4.4 Scalability

In order to evaluate the scalability of FTSP and LECTES, we evaluate the average, maximum and standard deviation of errors in multi hop network. We created a 9 hop network for FTSP and LECTES application. The evaluation setup of FTSP application is illustrated in Fig. 6.9. We modified the FTSP library to create a linear multi hop network, by specifying the destination of time synchronization packets (See Fig. 6.9(a)). In FTSP application, invoke node broadcasts a packet, and sensor nodes which received the packet forward it to the sink node with its reception time (See Fig. 6.9(b) and Fig. 6.9(c)). Note that the frequency of time synchronization is set to 0.8Hz and the frequency of invoke node transmitting packet is set to 1Hz. Fig. 6.10 shows the multi hop evaluation setup of LECTES application. In LECTES application, invoke node sends a packet to the specified target at 1Hz (See Fig. 6.10(a)). The



(a) Time synchronization (b) Invoke Node Broadcasts a Packet (c) Sender Nodes Forward the Packet to Sink Node

Figure 6.9: Scalability Evaluation of FTSP



(a) Invoke Node Sends a Packet to a Sender Node

(b) Sender Node Forwards the Packet to the Next Node

(c) Sender Node Forwards the Packet to the Next Node

Figure 6.10: Scalability Evaluation of LECTES

sender node which received a packet adds delay to the packet, and forwards it to the next node, and so on (See Fig. 6.10(b) and Fig. 6.10(c)). Note that the sink node snoops a packet whenever a sender node sends it to the next node. In FTSP application, average, maximum and standard deviation of errors are calculated according to the reception time on sender nodes. On the other hand, LECTES is evaluated by using the errors of the estimated time on the sink node. By setting up the evaluation environment as such, FTSP and LECTES are evaluated equally. The error is evaluated by comparing the data sent by each sender node. We conduct this evaluation for 48 hours in order to evaluate accurately.

Fig. 6.11 illustrates the average error of FTSP and LECTES in multi hop network. The x-axis indicates the number of hops, and the y-axis shows the average error in microseconds precision. The result shows LECTES always has smaller error compared to FTSP at any number of hops. As shown in the figure, the average error of FTSP increases drastically as the number of hops increases. On the other hand, that of LECTES also increases, however, the rate of increase is much smaller than that of FTSP. This shows that LECTES is superior to FTSP when it is used in multi hop WSNs.

Table 6.5 indicates the evaluation results of scalability. The average, maximum and standard deviation of errors are measured in a 9 hop network. Meanwhile, the packet loss rate is the average rate calculated in a single hop network. LECTES is better than FTSP in terms

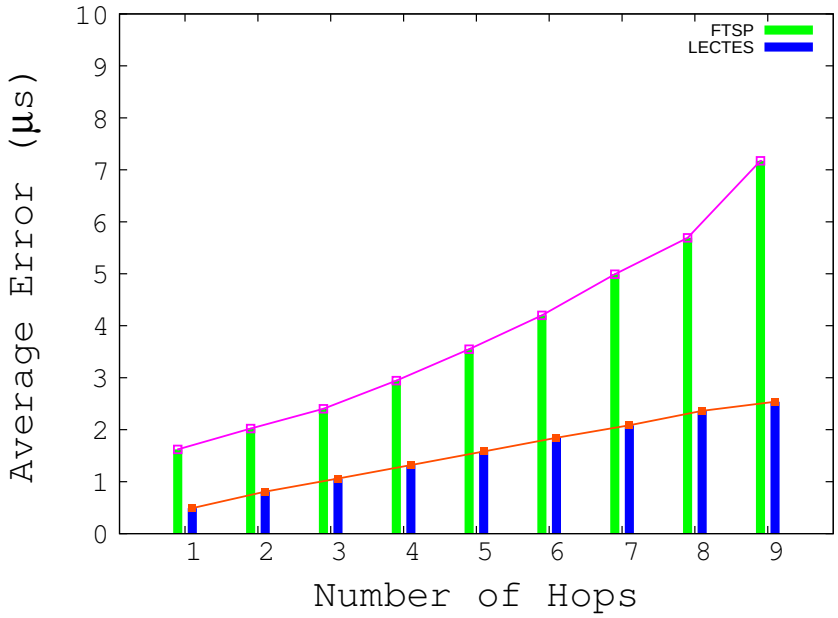


Figure 6.11: Average Error in Multi Hop Network

Table 6.5: Scalability Evaluation Results

		Simple Application	FTSP	LECTES
Scalability	Average Error	n/a	7.174 μs	2.538 μs
	Maximum Absolute Error	n/a	49 μs	21 μs
	Standard Deviation	n/a	2.052	1.173
	Packet Loss Rate	1.509%	2.851%	1.510%

of average, maximum and standard deviation of errors in a multi hop network. Although LECTES achieved higher accuracy than FTSP in a multi hop network, since the average error of LECTES is greater than 1 microsecond after 3 hops, some WSN applications such as counter sniper application [12] cannot work correctly.

Packet loss rate of simple application is 1.509%, probably because of the packet interference between other packets transferred over 2.4GHz wireless network. The packet loss rate of FTSP application came out to be 2.851%, while that of LECTES application became 1.510%. The packet loss rate of simple and LECTES application is equivalent, which shows the

superiority of LECTES to be used in WSN applications. Meanwhile, the packet loss rate has increased in FTSP application compared to that of simple and LECTES application. This, in turn, shows how FTSP is unfavorable in perspective of packet loss rate, which affects WSN application's robustness.

6.4.5 Storage Usage

Table 6.6 shows the size of compiled binary of each comparison target. We compiled the program that is used in the evaluation for Iris mote.

It is obvious that FTSP application requires huge size of data. This is because, FTSP application uses extra sending and receiving interfaces for synchronizing sensor nodes' clocks. The packet size of FTSP indicates the size of data packet (19 byte) and that of synchronization packet (13 byte). Since the frequency of synchronization is set to 0.8Hz in our evaluation, the packet size required by FTSP application is 29.4 ($= 19 \times 1.0 + 13 \times 0.8$) bytes per second. Note that the frequency of synchronization can be changed, however, lowering the frequency of it results in poor accuracy.

Storage usage of LECTES application is noteworthy. Application using LECTES uses 59 more bytes of RAM compared to the simple application, while reducing the ROM usage by 150 bytes and 294 bytes when delay information is 32 bit and 16 bit, respectively. When compared against FTSP application, LECTES application uses 8,102 or 8,246 less bytes of ROM and 452 less bytes of RAM, which is a significant difference. Since the storage size of

Table 6.6: Size of Compiled Binary for Iris Mote

	Simple Application	FTSP	LECTES(16 bit delay)	LECTES
ROM	14,366	22,320	14,218	14,074
RAM	540	1,011	559	559
Packet Size	19	19 & 13	15	13

Table 6.7: Information of Packets and Their Size

	Information	size (bit)	total size (byte)
Data Packet	Node ID	8	19
	Sequence Number	32	
	Voltage	32	
	Illuminance	16	
	Local Time	64	
FTSP Packet	Root Node ID	16	13
	Node ID	16	
	Sequence Number	8	
	Global Time	32	
	Local Time	32	
LECTES Packet	Node ID	8	15 OR 13
	Sequence Number	32	
	Voltage	32	
	Illuminance	16	
	Delay	32 OR 16	

sensor nodes are severely limited, the size of FTSP application is not preferable. On the other hand, LECTES application requires small storage size, which makes LECTES preferable to be used in WSNs.

Table 6.7 illustrates the information and its size contained in the packets. Simple application uses Data Packet, while FTSP application uses Data Packet and FTSP Packet. LECTES application uses LECTES Packet, which is the replacement of Data Packet. Note that the delay information of LECTES packet is stored by LECTES, not by the application, unless it is forcefully set through LectesInfo interface. Since FTSP requires FTSP packet other than data packet, the total size of exchanged packets is significantly large compared to that of the simple and LECTES application. LECTES successfully reduces the packet size by using the delay information instead of local time. Panthanchai et al. measured the amount of energy required to transmit packets with different payload size [23], and conclude that the bigger the size of packet's payload, the more energy required to transmit it. This,

in turn, shows that LECTES reduces the amount of energy required for transmitting packets compared against the simple application, while FTSP increases it. We consider the slight increase in sensor nodes' lifetime in LECTES is due to the smaller packet size.

The size of compiled FTSP application is significantly large compared to that of the simple application. Meanwhile, size of compiled LECTES application is considerably small; slightly more RAM and bit less ROM compared to that of the simple application. FTSP demands to exchange synchronization packets other than data packets, while LECTES reduces the size of data packets. For these reasons, we conclude that LECTES is superior to FTSP in perspective of storage usage, and packet size.

6.5 Summary

In this section, we summarize this chapter. First, we introduced the purpose of evaluation: checking the validity of LECTES as a solution for time precision problem. We then discussed the methodology of evaluation, including the evaluation environment, comparison targets and evaluation items. We chose three applications as comparison target; simple, FTSP and LECTES application. The evaluation procedure of each application is stated, and explained with pseudo code. Although the evaluation methodology is slightly different from each other, we assure that each application is evaluated fairly. Thirdly, we stated the evaluation items as follows; energy consumption, accuracy, scalability and storage usage. We precisely described the reasons we chose these items and the evaluation method. Finally, we explain the result of evaluation. The result of the whole evaluation is illustrated in Table 6.2. The evaluation proves that LECTES is superior to FTSP in terms of energy consumption, accuracy, scalability and storage usage. Despite the fact that LECTES is better than FTSP in terms of accuracy in multi hop network, we still have to improve the accuracy of LECTES in multi hop network, so that it can be adapted to greater number of WSN applications.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

In this thesis, we proposed LECTES, a scheme which solves the time precision problem in WSNs. In comparison to time synchronization protocol, LECTES does not synchronize sensor nodes' clocks, instead, it estimates the data acquirement time according to the receiver node's clock. Because of its characteristic, LECTES assumes the existence of sink node in WSNs. We addressed the issues of current time synchronization protocols as follows; energy consumption, scalability, storage usage and clock adjustment method. LECTES improves these factors, especially the energy consumption, which is one of the most important factors in wireless sensor nodes, because it affects the lifetime of them.

We implemented LECTES as a TinyOS library, and evaluated it on Iris mote. We proposed four requirements of LECTES as follows; low energy consumption, high accuracy, high scalability and small storage usage. Resources of the wireless sensor nodes are severely restricted as a trade off for being small and cheap, thus, it is important to minimize the amount of resource usage. LECTES is implemented in MAC layer to increase the accuracy while minimizing the energy consumption. Sensor nodes using LECTES store or add delay to the packet. We define delay as the time required to handle data or packets. The receiver node subtracts the delay from the reception time of the packet to estimate the data acquirement

time.

We evaluated LECTES and compared it against simple application and FTSP [19]. The simple application acquires illuminance value and sends packets at a specific frequency. FTSP is a time synchronization protocol, which is said to be achieving high accuracy with considerably small amount of energy. The evaluation results (See Table 6.2) prove that LECTES is superior to FTSP in terms of energy consumption, accuracy, scalability and storage usage. This shows the effectiveness of using LECTES instead of time synchronization protocols. LECTES is implemented in MAC layer of mote, thus, it is unable to adapt to the wireless sensor nodes whose MAC layer is not implemented with software. Note that when using LECTES, sensor nodes' clocks are not synchronized. Therefore, LECTES cannot be used in an application where sensor nodes store sensing data on their own storage.

7.2 Future Work

This section introduces the future work of LECTES. We first mention the compatibility of LECTES followed by the improvement in scalability of LECTES.

We implemented LECTES as a TinyOS library for Iris mote. However, Iris mote is only a part of mote family, and great number of other kinds of motes are used all over the world. Given this fact, we want to provide LECTES not only for Iris mote, but also for other motes, such as Mica2, Micaz and Telos mote. Users or application providers should be able to use LECTES library without any stress, therefore, we should consider using LECTES with other useful implementation, such as CTP, which is used to create a multi hop network. There are many other kinds of useful implementations, and allowing the use of LECTES with them would increase the variety of WSN applications. Therefore, we need to make sure the use of LECTES with other implementations would have no side effects.

Through the evaluation, we confirmed that LECTES is superior to current time synchro-

nization protocols. However, we concern about the scalability of LECTES, because, in the near future, there would be large scale WSNs consisting of hundreds of thousands of sensor nodes. For example, Kim et al. created a 46 hop network with 64 mote on a bridge to monitor its healthiness [15]. According to the result of scalability evaluation, we confirmed the increase in the error as the number of hops increases. The rate of the increase in the error is smaller in LECTES compared to FTSP, however, we should maintain the rate of increase as small as possible, so that it can be adapted to greater number of WSN applications.

Acknowledgments

First and foremost, I would like to thank my advisor, Professor Hideyuki Tokuda for his technical and professional advice, guidance and encouragement.

I would like to thank Lecturer Jin Nakazawa for his valuable comments and encouragement to this thesis and on my research. I would like to thank Professor Jun Murai, Associate Professor Hiroyuki Kusumoto, Professor Osamu Nakamura, Associate Professor Kazunori Takashio, Assistant Professor Noriyuki Shigechika, Assistant Professor Rodney D. Van Meter III, Associate Professor Keisuke Uehara, Associate Professor Jin Mitsugi and Professor Keiji Takeda for valuable and technical comments on this thesis.

I am extremely thankful for Dr. Takuro Yonezawa, who took care of me since I joined Hide. Tokuda Lab., for his great support and advice. I would also like to thank seniors of move! research group, Dr. Hiroshi Sakakibara, Dr. Michio Honda, Mr. Naoki Nakagawa, Ms. Thi-huong-giang Vu, Mr. Takahiro Nozawa and Mr. Takayoshi Araki for their comments and advice on my research. I am grateful to Mr. Takahiro Nozawa, one of a master degree student of move! research group, for treating me as if I am his brother. I thank numerous members of Tokuda and Murai Laboratory for their great support.

Last, but not least, I appreciate my beloved parents Hiroyuki and Masae, adorable sisters Konomi and Chihiro, and my brother grape for the consecutive support of my daily life.

February 12, 2010

Kenji Yonekawa

Bibliography

[1] A. Agarwal and S. Das. Dead reckoning in mobile ad hoc networks. In *Wireless Communications and Networking, WCNC, IEEE*, volume 3, pages 1838–1843, 2003.

[2] Atmel. Atmega1281. http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf, Nov 2009.

[3] Atmel. Atmega128L. http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf, Nov 2009.

[4] Atmel. RF230. http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf, Dec 2009.

[5] M. Beigl, C. Decker, A. Krohn, T. Riedel, and T. Zimmer. μparts: Low cost sensor networks at scale. *Ubicomp: The Seventh International Conference on Ubiquitous Computing*, 2005.

[6] Crossbow. Iris. http://www.xbow.com/Products/productdetails.aspx?sid=264, Nov 2009.

[7] Crossbow. Mote. http://www.xbow.com/Products/productdetails.aspx?sid=156, Nov 2009.

[8] Inc. Crossbow Technology. Crossbow Technology. http://www.xbow.com/, Nov 2009.

[9] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review*, 36(SI):147–163, 2002.

[10] S. Ganeriwal, R. Kumar, and M. Srivastava. Timing-sync protocol for sensor networks. In *SenSys: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149. ACM, 2003.

[11] C. Gui and P. Mohapatra. Power conservation and quality of surveillance in target tracking sensor networks. In *MobiCom: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 129–143. ACM, 2004.

[12] S. Gyula, M. Maróti, A. Ákos, B. György, K. Branislav, A Nádas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *SenSys: In Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 1–12. ACM, 2004.

[13] M. Horauer, K. Schossmaier, U. Schmid, and N. Kerö. Psynutc-evaluation of a high precision time synchronization prototype system for ethernet lans. In *in Proceedings of 34th Annual Precise Time and Time Interval Meeting (PTTI)*, pages 263–279, 2002.

[14] Texas Instruments. CC2420. http://focus.ti.com/lit/ds/symlink/cc2420.pdf, Dec 2009.

[15] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *IPSN: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 254–263. ACM, 2007.

[16] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real-time systems. pages 514–521, 1989.

[17] A. Lédeczi, A. Nádas, P. Völgyesi, G. Balogh, B. Kusy, J. Sallai, G. Pap, S. Dóra, K. Molnár, M. Maróti, and G. Simon. Countersniper system for urban warfare. *Transactions on Sensor Networks*, 1(2):153–177, 2005.

[18] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97. ACM, 2002.

[19] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The flooding time synchronization protocol. In *SenSys: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49. ACM, 2004.

[20] Sun Microsystems. SunSPOT. http://www.sunspotworld.com/, Nov 2009.

[21] D. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39:1482–1493, 1991.

[22] M. Mock, R. Frings, E. Nett, and S. Trikaliotis. Continuous clock synchronization in wireless real-time applications. *Reliable Distributed Systems, IEEE Symposium on*, 0:125, 2000.

[23] Y. Panthanchai and P. Keeratiwintakorn. An energy model for transmission in telos-based wireless sensor networks. *International Joint Conference on Computer Science & Software Engineering*, 2007.

[24] M. Qun and M. Daniela. Global clock synchronization in sensor networks. *IEEE Trans. Comput.*, 55(2):214–226, 2006.

[25] V. Shnayder, M. Hempstead, B. Chen, W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *SenSys: Proceedings of the*

2nd international conference on Embedded networked sensor systems, pages 188–200. ACM, 2004.

[26] W. Su and I. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 13(2):384–397, 2005.

[27] TinyOS. TinyOS Community Forum. http://www.tinyos.net/, Nov 2009.

[28] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh. Deploying a wireless sensor network on an active volcano. *Internet Computing, IEEE*, 10(2):18–25, 2006.

[29] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *SenSys: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 13–24. ACM, 2004.

[30] Y. Zeng, B. Hu, and S. Liu. Vector kalman filter using multiple parents for time synchronization in multi-hop sensor networks. In *IEEE Sensor and Ad Hoc Communications and Networks Conference (SECON)*, pages 413–421, 2008.