CHALMERS

Clock Synchronization in Sensor Networks for Civil Security

Master of Science Thesis in the Programme Networks and Distributed Systems

FARNAZ MORADI
ASRIN JAVAHERI

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, March 2009

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.
The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Clock Synchronization in Sensor Networks for Civil Security

FARNAZ. MORADI,
ASRIN. JAVAHERI,

© FARNAZ. MORADI, March 2009.
© ASRIN. JAVAHERI, March 2009.

Examiner: PHILIPPAS TSIGAS

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden March 2009

ABSTRACT

Wireless sensor networks are widely deployed in security surveillance applications. Since most of these applications depend on having a common notion of time among the sensors, performing fine-grained clock synchronization is highly desirable. Most of the existing time synchronization approaches mainly focus on improving clock precisions and reducing energy consumption, while ignoring the effect of faults and attacks on system performance. In this thesis we show the importance of fine-grained clock synchronization and implement two of the most well-known synchronization schemes proposed in the literature. We compare these approaches considering the precision, cost and fault tolerance. Our implementations tolerate node failures and adopt newly joint nodes to the network. We also use the synchronized time to avoid message collisions by emulating TDMA-based scheduling in the synchronization protocol. Finally we describe some of the existing attacks against clock synchronization protocols and some of the possible solutions.

ACKNOWLEDGEMENT

We would like to express our gratitude to all those who gave us the possibility to complete this thesis. First and foremost we are deeply indebted to our thesis advisor Philippas Tsigas from Chalmers University of Technology for trusting us and offering the chance to work on this thesis. His help, stimulating suggestions, constructive comments and encouragements have greatly improved this work.

We would like to express our appreciation and thank to our supervisor Pablo Suarez from Saab Security in Kista, for all his help, support, interest and valuable hints.

We are also grateful to Mohamad Reza Shoaei for his contribution in the work which assisted us in numerous ways.

We should also thank people from NES group at SICS who helped us overcome implementation problems we faced. Also we are thankful to Andreas Larson for the time he spent on reviewing our work.

And last, but most importantly, both of us are deeply grateful to our families, whose love and guidance is always with us in whatever we pursue. It is to them that we dedicate this work.

CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

Advances in digital electronics technology have led to reductions in size, computing capacity, power consumption, and cost of computing devices. According to Moore's law, the number of transistors in an integrated circuit increases exponentially, doubling approximately every two years. These miniaturized computing devices together with enhancements in micro-electro-mechanical systems technology and wireless communications, raised the idea of developing small, low-cost, low-power, multifunctional wireless sensor nodes [1, 2, 3, 29, 34]. History of development of sensor nodes dates back to 1998 in Smart Dust[1] project in University of California. Although this project finished early, but many more research projects have grown out of it for creating sensor-rich "smart environments" [13].

These tiny individual sensor nodes are very resource constrained and have limited processing speed, storage capacity, and communication bandwidth. These devices have substantial processing capability when aggregated with each other in a distributed manner to form a wireless sensor network (WSN). These nodes can be distributed in large scales and deployed in unattended environments to monitor and sense local conditions, process and communicate gathered information, and perform coordinated actions with other nodes.

The creation of large-scale wireless sensor networks by interconnecting several hundred or thousands of sensor nodes have found many potential applications in military, environmental, medical and civilian domains. Each of these applications has their own requirements and constraints such as price, size or battery lifetime of the nodes that must be taken into account in order

[1] http://robotics.eecs.berkeley.edu/~pister/SmartDust/

to develop successful products and services. For instance in civilian domains, which are the topic of interest in this thesis, employing reliable and robust sensor network for monitoring and security surveillance is very attractive. In such applications, data from each sensor should be aggregated using data fusion to form a single meaningful result [1]. This mandates establishing a common, highly accurate time frame across nodes [34]. This is especially critical in applications that depend on a global notation of time in the whole network, such as mobile object tracking. Therefore, performing clock synchronization is highly desirable and has attracted considerable research attention in recent years.

1-1 Problem Statement

One of the basic middleware services of sensor networks is time synchronization [4]. Precisely synchronized clocks are more important for sensor networks than traditional centralized or Internet-based applications. One example that illustrates the need of precise clock synchronization is the formation of a TDMA schedule for low-energy radio operation. This is an important application because listening and transmitting are both very energy-expensive operations in a low-power radio and one of the most important constraints on sensor nodes is the low power consumption requirement. Fine grained clock synchronization is crucial for efficient TDMA radio scheduling among sensor nodes to allow nodes turn they radio off to conserve energy.

Other examples that require fine-grained time synchronization are: measuring the time-of-flight of sound, distributing a beam forming array, or suppress redundant messages by recognizing duplicates of the same event by different sensors [3]. In addition to these domain-specific requirements, sensor network applications often rely on synchronization for secure cryptographic schemes, coordination of future actions, ordering logged events during system debugging to mention a few of such applications.

There are many clock synchronization approaches proposed in the literature [2, 3, 4, 9, 16]. Many of these approaches can provide fine-grained synchronization for sensor networks but they are usually focused on synchronization precision and energy efficiency while ignoring effect of faults or failures during the sensor network lifetime. For many applications such as security surveillance applications, it is necessary to have a robust clock synchronization that can tolerate failures and mask attacks in unattained environments.

1-2 METHOD

In this master thesis we briefly review some of the clock synchronization algorithms proposed in the literature and select two of the most well-known algorithms for implementation and comparison. The selected algorithms are implemented on Contiki operating system [6] and tested on MSB-430 [5] platform. Since one of the most important requirements of synchronization services is robustness, we analyze the selected algorithms in presence of faults such as node failures. We also reviewe some of the possible security attacks that can be launched to tamper with time synchronization and implemente a secure and self-stabilizing algorithm proposed in [8].

1-3 CONTRIBUTION

In this thesis we review some of the clock synchronization algorithms proposed in the literature and analysis two different approaches by implementing and testing them on our MSB-430 platform. The first approach is simple and centralized which synchronizes a set of nodes to a leader. The second approach is distributed and more complex and can synchronize a set of nodes with each other. We compare performance of these approaches considering precision and cost. To our best knowledge, this is the first time that these synchronization algorithms are implemented on MSB-430 nodes using the Contiki operating system.

Our implementation tolerates failures that may occur due to message collisions and node failures and (re)joins. We implemented a leader election mechanism to compensate for synchronizer node failure in the first approach. We also implement a random back-off strategy to reduce the number of message losses and a TDMA-based scheduling for avoiding collisions in the second approach. There are trade-offs between fine-grained synchronization and energy efficiency, and between reliability and complexity. Finally we show how these implementations can be employed in a real security surveillance application and how faults and failures are tolerated

1-4 LIMITATIONS

Due to the large number of variations of clock synchronization algorithms for sensor networks, only a limited number of them are mentioned here and only two algorithms are implemented. Due to hardware constraints clock synchronization precision is less than what we can actually achieve with these

algorithms on an ideal platform. Given enough time, further optimizations could improve the robustness and precision of certain parts of the implementations.

1-5 STRUCTURE

The remaining part of the thesis is structured as followed. In chapter 2 we briefly review the importance and requirements of clock synchronization algorithms and briefly describe some of the protocols proposed in the literature especially RBS [3] and FTSP [4] protocols. We also describe the notations and definitions required for understanding the algorithms, and finally introduce the Contiki operating system and the MSB-430 platform used for the implementations. Chapter 3 covers the design and implementation choices, followed by the evaluation of implementation results in chapter 4. In chapter 5 we describe the fault-tolerance issues in clock synchronization algorithms and propose some of the possible countermeasures. In chapter 6 we review security problems in synchronization algorithms followed by a description of a secure and self-stabilizing clock synchronization algorithm. Chapter 7 presents a commonly used sensor network application which represents the importance of having robust clock synchronization. Finally, the thesis is concluded in chapter 8 and some of the possible future works are suggested.

Chapter 2

BACKGROUND

This chapter introduces the importance of clock synchronization in sensor networks and describes the requirements of synchronization protocols. Some of the notations and definitions necessary for understanding the clock synchronization algorithms are described. We also review some of the existing algorithms for fine-grained clock synchronization. Finally we introduce the Contiki operating system and the MSB-430 platform which is used for the implementation of selected algorithms.

2-1 CLOCK SYNCHRONIZATION

Clock synchronization has been the focus of many researches over years, and many time synchronization algorithms have been proposed so far.

Clock synchronization is the process of ensuring that physically distributed processors have a common notion of time [2]. In centralized systems each process can get the time by issuing a system call to the kernel, so there is no time ambiguity between different processes. Distributed systems, in contrast have no global clock or shared memory and each processor has its own local clock. These clocks drift away over time and pose problems to applications that depend on synchronized clocks. This clarifies the importance of implementing precise clock synchronization protocols in distributed systems such as sensor networks. The requirements of such protocols as listed in [2] are as follows:

- The protocol should cope with unreliable message transmission and unbounded message latencies.
- Nodes getting synchronized must be able to estimate the local time on the other node's clock.

- Time must run forward over time.
- Synchronization overhead must not degrade system performance.

Clock synchronization algorithms proposed in the literature can be classified according to different criteria such as sender-to-receiver, receiver-to-receiver, master-slave, peer-to-peer, single-hop and multi-hop synchronization and so on. A comprehensive classification of synchronization protocols is described in [2]. Here we briefly state some of the rich approaches for design of clock synchronization algorithms as described in [9].

- *Leader-based clock synchronization*. Clocks of all nodes get synchronized to one leader clock. Each node periodically transmits a time-stamped beacon. Upon reception of the beacon, each node except the leader copies the time-stamp to its clock.
- *Pulse-based clock synchronization*. This method has biological inspiration and could be promising for pulse-coded radio protocols.
- *Reference broadcast clock synchronization*. A repeatedly transmitted pulse signal is recorded simultaneously at all nodes, and subsequent conversation among receivers results in consensus for the global time.
- *Averaging-based clock synchronization*. Each node periodically transmits a time-stamped beacon, and each node adjusts its clock to be the average of its neighbors.
- *Converge to max clock synchronization*. Each node periodically transmits a time-stamped beacon. Upon reception of the beacon, the node adjusts its clock to agree with the beacon only if the timestamp is greater than its global clock.

Later in this section we describe a leader-based (FTSP) and a reference broadcast (RBS) clock synchronization algorithm.

Since clocks in distributed systems are not ideal and do not run with exact same frequencies, they drift away over time. This mandates that clock synchronization protocol be executed continuously to re-synchronize the nodes. The execution of a clock synchronization protocol can be classified to on-demand (also known as post-facto [3]) synchronization and continuous synchronization [8].

In on-demand synchronization, nodes can keep their clocks unsynchronized, and only run a distributed procedure for clock synchronization after a particular event occurs. In this way nodes can stay in a low power state in times when synchronized clocks are not required. When nodes' clocks reach the necessary precision, the synchronization procedure can be stopped. In continuous synchronization, the procedure of clock synchronization is never stopped. Continuous procedure guarantees fine-grained clock synchronization. Therefore, there is a trade-off between precision requirements of applications and energy constraints of sensor nodes. In this

thesis we consider continuous fine-grained synchronization for our implementations.

2-1-1 CLOCK AND TIME

Sensor network applications need clocks to measure elapsed time, schedule tasks and compare time of sensor readings in different nodes. A computer clock is an electronic device that counts oscillations in a quartz crystal, at a particular frequency [2]. These clocks are essentially timers. The timer counts the oscillations of the crystal, which is associated with a counter register and a holding register. For each oscillation in the crystal, the counter is decremented by one. When the counter becomes zero, an interrupt is generated and the counter is reloaded from the holding register. Therefore, it is possible to program a timer to generate an interrupt by setting an appropriate value in the holding register, where each interrupt is called a clock tick. At each clock tick, the interrupt procedure increments the clock value stored in memory [2].

Here we follow the clock notations which are compatible with that of [9]. The operating system encapsulates the hardware counter by a software module called the *Native Clock* and the native time is obtained from that. The native clock is encapsulated by a module called the *Local Clock* which can increment for a longer period without rollover. And finally, one more layer of encapsulation creates the *Global Clock* module. The clock synchronization algorithms never adjust native clock but adjust the local clock to achieve precise synchronized global time.

There are different reasons why nodes represent different times in their respective clocks. The nodes might have been started at different times introducing arbitrary phase offsets. Since clock counters do not increment at ideal rates the quartz crystals at each of the nodes might be running at slightly different frequencies, causing the clock values to gradually diverge from each other. Finally, the frequency of the clocks can change variably over time because of aging or ambient conditions such as temperature.

Different definitions related to clocks and times are given in Figure 1. Clock *offset* is defined as the difference between the time reported by a clock and the real time. The first derivation of the clock offset value with respect to real time is known as *skew*. The skew of a clock is the difference in the frequencies of the clock and the perfect clock. The second derivation of clock's offset with respect to time is called *drift*.

Time: The time of a clock in a processor p is given by the function $C_p(t)$, where $C_p(t) = t$ for a perfect clock.

Frequency: Frequency is the rate at which a clock progresses. The frequency at time t of clock C_a is $C'_a(t)$.

Offset: Clock offset is the difference between the time reported by a clock and the real time. The offset of the clock C_a is given by $C_a(t) - t$. The offset of clock C_a relative to C_b at time $t \geq 0$ is given by $C_a(t) - C_b(t)$.

Skew: The skew of a clock is the difference in the frequencies of the clock and the perfect clock. The skew of a clock C_a relative to clock C_b at time t is $(C'_a(t) - C'_b(t))$.

If the skew is bounded by ρ, clock values are allowed to diverge at a rate in the range of $1-\rho$ to $1+\rho$.

Drift (rate): The drift of clock C_a is the second derivative of the clock value with respect to time, namely $C''_a(t)$. The drift of clock Ca relative to clock C_b at time t is $(C''_a(t) - C''_b(t))$.

Figure 1Clock terminology [2].

2-1-2 TIME SYNCHRONIZATION ERRORS

Non-deterministic delays in message deliveries in wireless sensor networks can adversely affect the required precision of clock synchronization. These delays that contribute directly to synchronization errors need to be carefully analyzed and compensated for. Sources of the message delivery delays has been first introduced and characterized in [10] as having four distinct components:

1. *Send Time*. The time spent at the sender to construct the message. This time includes kernel protocol processing and variable delays introduced by the operating system, and the time used to issue the send request to the network interface of the sender. Depending on the system call overhead of the operating system and on the load of the processor, the send time can be as high as hundreds of milliseconds.
2. *Access Time*. Delay incurred waiting for access to the radio channel before the transmission begins. This is specific to the MAC protocol in use. The access time is the least deterministic part of the message delivery and can vary from milliseconds up to seconds.
3. *Propagation Time*. The time it takes for the message to travel from sender to receiver once it has left the sender. When both the sender and receiver share access to the same physical media, this time is simply the physical propagation time of the signal through the media and is very small. The propagation time is highly deterministic and only depends on the distance between the sender and receiver. This time is less than one microsecond for ranges under 300 meters [4].

4. *Receive Time.* The time it takes for the receiver to process the incoming message by network interface to receive the message and notify the application of its arrival. The characteristics of receive time are similar to that of the send time. But if the arrival time is time stamped at a low enough level in the operating system kernel, the receive time does not include the overhead of system calls, context switches, or transfer of message from network interface to the application.

In [4] more specific sources of message delivery delay errors are introduced:

5. *Interrupt Handling Time.* The delay incurred between the time when the radio chip raises the interrupt and the microcontroller responds to it, which is mostly less than a few microseconds.
6. *Encoding Time*. The time it takes for the radio chip of the transmitter to encode a message to electromagnetic waves. This time is in the order of a hundred microseconds.
7. *Decoding Time*. The time it takes for the radio chip of the receiver decode the received electromagnetic waves to the message data. This time is also in the order of hundred microseconds.
8. *Byte Alignment Time*. The delay introduced by different byte alignment of the sender and receiver. This time can be computed from the bit offset and the speed of the radio [4].

Many of the existing time synchronization algorithms use different methods for estimating and compensating for these sources of errors. But in some schemes, approaches to remove the source of errors from the critical path are used to reduce errors.

2-1-3 Approaches to Time Synchronization

Applications may need to know the exact time of the day when an event happens. These applications mainly use an external timescale for synchronization which is typically provided by the Global Positioning System (GPS). Commercial GPS receivers can achieve accuracy of better than 200nsec relative to Coordinated Universal Time (UTC) [3, 11], but usually are too expensive to be used on cheap sensor nodes. GPS requires a clear sky view which is not available for indoor scenarios, and is costly and high-power to be employed on an energy constrained sensor node [31].

The Network Time Protocol (NTP) [12] is perhaps one of the most advanced time synchronization protocols which is now the de-facto standard for time synchronization on the Internet. The design of NTP involves a hierarchical tree of time servers. The root server synchronizes with the UTC and the next level servers act as a backup to the root server. The clients are at the lowest

level of synchronization subnet. Unfortunately many of the assumptions that NTP makes are not true in domain of sensor networks, because the time synchronization requirements in the context of sensor networks differs significantly.

The combination of GPS and NTP has proved very successful [13]. But they are not suitable for use in wireless sensor networks because of complexity, cost, scalability and energy issues [31].

Many applications in distributed systems need to know the relative ordering of events that happened on different processors. Lamport's work which is a landmark in computer clock synchronization clarified the importance of virtual clocks in systems where causality is more important than the absolute time [14]. Lamport's had an important influence in sensor networks since many sensor applications require only relative time rather than absolute time [13].

Cristian [15] proposed a probabilistic synchronization method that exploits a large number of messages to get the accurate shortest round-trip time with high probability. This method is used by many other clock synchronizing protocols which a process sends a time request and waits for the remote process to respond. Upon receiving the response, the process calculates the round-trip as the difference between the time at which it initiated the request and the time at which it received the response [2].

The TPSN [16] algorithm uses the conventional approach of sender receiver synchronization. In this algorithm a hierarchical structure is established in the network and then a pair wise synchronization is performed along the edges of this structure. Each node gets synchronized by exchanging two synchronization messages with its reference node one level higher in the hierarchy. Eventually all nodes in the network synchronize their clocks to a reference node. TPSN achieves good performance by time-stamping the messages in the Medium Access Control (MAC) layer of the radio stack.

There are many other synchronization schemes presented in the literature such as TSS, Tiny-Sync, LTS, TSync, AD, TDP and so on. A short description of each of these algorithms can be found in [2, 31].

For our implementations, we decided to select a reference broadcast synchronization method called RBS [3] which is based on receiver-to-receiver synchronization scheme, and a leader-based synchronization protocol named FTSP [4] that is a sender-to-receiver synchronization protocol. These protocols are described with more details in the following sections.

Almost all of the aforementioned clock synchronization algorithms do not take security, failures and message interferences into account. In recent years many secure clock synchronization approaches are introduced. In [32] a

secure time synchronization toolbox for securing pair wise sender-to-receiver time synchronization in sensor networks. In [33] nodes use redundant ways for synchronizing their clocks to a common source, so that they can tolerate false or missing synchronization information send by compromised nodes.

However, most of these proposed secure clock synchronization approaches are not self-stabilizing [8]. Self-stabilizing algorithms can tolerate transient faults [30]. After the occurrence of the transient fault, a self-stabilizing algorithm converges the system to a global consistent state to finish its task. More about self-stabilizing clock synchronization can be found in [30]. In [8] the first secure and self-stabilizing clock synchronization algorithm is presented that provides fine-grained synchronization is presence of compromised nodes. For this thesis we have selected this algorithm to review with more details in chapter 6 (see section 6-2).

2-1-4 Reference Broadcast Synchronization

Reference Broadcast Synchronization (RBS) [3, 13], synchronizes a set of receivers with one another. In this method nodes broadcast reference beacons to their neighbors. A reference broadcast does not contain an explicit timestamp; instead, receivers use its arrival time as a point of reference for comparing their clocks.

The main advantage of RBS is that a broadcast message is received almost concurrently (even though its delay is largely variable), and thus the synchronization error typically is smaller than with unidirectional or round-trip synchronization.

The simplest form of RBS is the broadcast of a single pulse to two receivers, allowing them to estimate their relative pulse offsets:

1. A transmitter broadcasts a reference packet to two receivers
2. Each receiver records the time that the reference was received, according to its local clock.
3. The receivers exchange their observations.

According to [3] the precision of synchronization can increase by sending more than one reference:

1. A transmitter broadcasts *m* reference packets.
2. Each of the *n* receivers records the time that the reference was observed, according to its local clock.
3. The receivers exchange their observations.

4. Each receiver I can compute its phase offset to any other receiver j as the average of the phase offsets implied by each pulse received by both nodes I and j. That is, given
   n: the number of receivers
   m: the number of reference broadcasts, and
   $T_{r,b}$: r's clock when it received broadcast b,
   $$\forall i \in n, j \in n: Offset[i,j] = \frac{1}{m}\sum_{k=1}^{m}\left(T_{j,k} - T_{i,k}\right).$$

This basic scheme does not account for clock skew, so instead of averaging the phase offsets for multiple observations, RBS performs a least-squares linear regression. This offers a fast, closed-form method for finding the best fit line through the phase error observations over time. The frequency and phase of the local clock of the nodes with respect to the remote node can be recovered from the slope of the line and its intercept with the y axis [3].

The fundamental property of RBS is that a broadcast message is only used to synchronize a set of receivers with one another. Doing so eliminates the Send time and Access time from the critical path. The send time and access time are typically the largest source of error and biggest contributors to the none-determinism in the latency. Also with minimal operating system modification to read the clock at interrupt time, the Receive time can become much shorter. Therefore the critical path length in RBS only includes the time from injection of the packet into the channel to the last clock read. As depicted in Figure 2 RBS is only sensitive to the difference in propagation time between a pair of receivers.

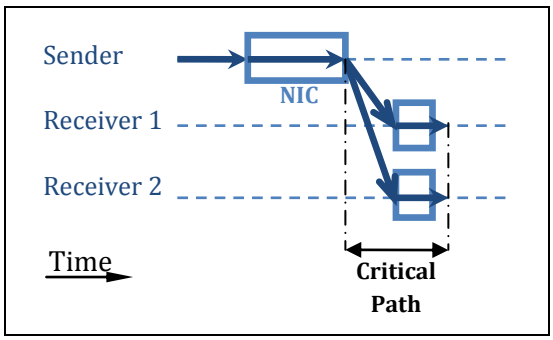

Figure 2 Critical path analysis for RBS

2-1-5 FLOODING TIME SYNCHRONIZATION PROTOCOL

Flooding Time Synchronization Protocol (FTSP) [4] utilizes MAC-layer time-stamping and compensates for errors including clock skew estimation. This protocol presents some techniques for mitigating effects of delays and other sources of uncertainties in message transmission. The FTSP achieves robustness by utilizing periodic flooding of synchronization messages.

The FTSP take advantage of radio broadcast to synchronize multiple receivers to the time provided by the broadcast. The broadcasted message contains the sender's time stamp which is the estimated global time at the transmission of a given byte. Receivers upon the receipt of the broadcast read their local time and use the difference between the global and the local time pair to estimates their clock offset. Time-stamping on the sender side is done in MAC layer before the bytes containing the time stamp are transmitted. The FTSP time-stamping effectively reduces the jitter of the interrupt handling and encoding/decoding times by recording multiple time stamps both on the sender and receiver sides. The time stamps are made at each byte boundary after the SYNC bytes as they are transmitted or received [4].

The basic FTSP algorithm is as follows:

1. A transmitter broadcasts *m* messages time-stamped with its estimated global time.
2. Each of the *n* receivers obtains the corresponding local time from their respective local clock at message reception.
3. Each receiver estimates the skew and offset of its local clock from that of transmitter using linear regression on the past *m* data points.

The proposed FTSP algorithm in [4] uses a fine-grained clock, MAC layer time-stamping with several jitter reduction techniques to achieve high precision. FTSP utilizes less network resources than RBS, and eliminates the Send and Access time errors (containing interrupt handling and encoding times) in the sender side, and removes the Receive time error (including decoding, byte alignment and interrupt handling times) in the receiver, but does not compensate for the Propagation time. Figure 3 demonstrates the decomposition of message delivery delays. The dots represent the times when messages crosses each of the presented layers: software (cpu), hardware (radio chip) and physical (antenna). The triangles represent the times when the time-stamping is done. Depending on the hardware the time-stamping is usually done by the microcontroller when it handles the radio chip interrupts in the radio driver of the transmitter and the receiver.

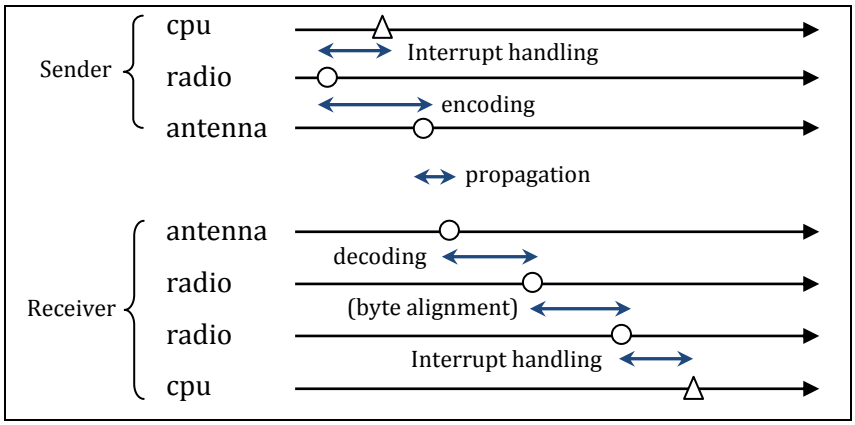

Figure 3 The timing of the transmission of an idealized point in different layers of the sender and the receiver [4].

2-2 SENSOR NETWORK PLATFORM

In this section the platform used for implementation of clock synchronization algorithms is described briefly.

2-2-1 MODULAR SENSOR BOARD

Modular Sensor Board (MSB) [5] is chosen as the platform for implementing clock synchronization. MSB nodes' layout and peripherals are designed to fit research needs of the near future. The modular structure of these nodes allows use of different modules for energy supply and sensing for different purposes.

The core module of MSB is a complete sensor node, which contains a microcontroller (MCU), radio transceiver, external storage and two sensors for humidity and temperature sensing. Figure 4 shows the core module of MSB-430 which is 36×41mm large.

The Texas Instrument MSP430x1xx-series MCU offers 60 KB of memory divided into 5 KB RAM and 55 KB Flash-ROM. It is clocked by a digital controlled oscillator (DCO) which can be configured from software between 1 to 11MHz. For synchronization the external 32.768 kHz quartz is used. Instead of EEPROM an SD-/MM-card slot is included for secondary storage of up to 4 GB (32 GB with SDHC). It is connected to a UART and accessed using the SPI protocol.

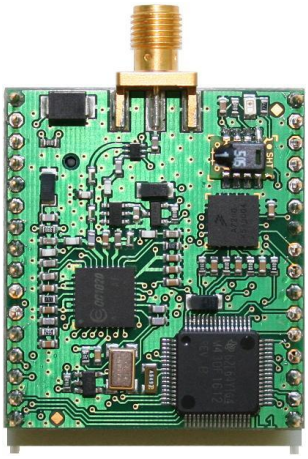

Figure 4 MSB-430 Core Module

The MSB-430 radio uses the license free 868 MHz ISM band. A Chipcon CC1020 transceiver [17] is used in combination with an additional low-noise amplifier on the receiver. The radio frequency can be selected separately for receiver and transmitter by software. This allows usage of multiple radio channels for advanced routing schemes. Transmission power can also be adjusted to reduce power consumption.

2-2-2 CONTIKI OPERATING SYSTEM

The open-source operating system, Contiki [6], is a lightweight, highly portable, multi-tasking operating system that can be ported to many platforms such as MSB-430. Contiki is designed for embedded systems with small amounts of memory. A typical Contiki configuration is 2KB of RAM and 40KB of ROM. The hardware-independent part of Contiki is written in the C programming language. The system is designed to be portable and has been ported to a number of microcontroller architectures, including the Texas Instruments MSP430 and the Atmel AVR [18].

Contiki has the ability to load and unload individual applications or services at run-time instead of a complete binary image of the entire system and therefore requires less energy and less time when transmitting an application through a network [18].

The Contiki system design is based on an event-driven execution model which is often used in operating systems designed for resource-constrained environments. In event-driven systems, processes are implemented as event handlers which cannot be preempted by other processes; therefore, all processes can use the same stack to effectively share the scarce memory resources. However, in a purely event-driven operating system a lengthy

computation completely monopolizes the CPU and unable the system to respond to external events.

In preemptive multi threaded operating systems, on the other hand, lengthy computation could be preempted. However, each thread must have its own stack and the memory contained in a stack cannot be shared between many concurrent threads.

To combine the benefits of both event-driven systems and preemptible threads, Contiki uses a hybrid model in which Contiki processes use light-weight protothreads [19] that provide a linear, thread-like programming style on top of the event-driven kernel. Contiki also supports per-process optional preemptive multi-threading, and inter-process communication using message passing through events.

2-2-2-1 Communication stacks

Contiki operating system contains two communication stacks: uIP and Rime. uIP is a RFC-compliant TCP/IP stack with reduced functionalities that allow Contiki to communicate over the Internet. Rime is a lightweight stack that allows defining a number of functionalities combining different primitives. A brief summary of both stacks is presented next.

2-2-2-1-1 uIP

The uIP [20] communication stack was designed to allow small, memory-constrained 8-bit micro-controller based systems to communicate using the TCP/IP suite. The uIP stack only supports the minimal requirements for a TCP/IP basic communication. It includes reduced versions of IP, ICMP, UDP and TCP protocols. Enabling Internet connectivity using these protocols allows a number of new applications and the improvement of existing ones.

2-2-2-1-2 Rime

Rime [7] is a lightweight layered communication stack designed for low-power radios particularly for wireless sensor networks. Rime is organized in layers as shown in Figure 5, where the more complex protocols are implemented using the less complex protocols. The layers are designed to be extremely simple. Since the layers are simple and light and each requires a very small header, codes can be reuse different levels.

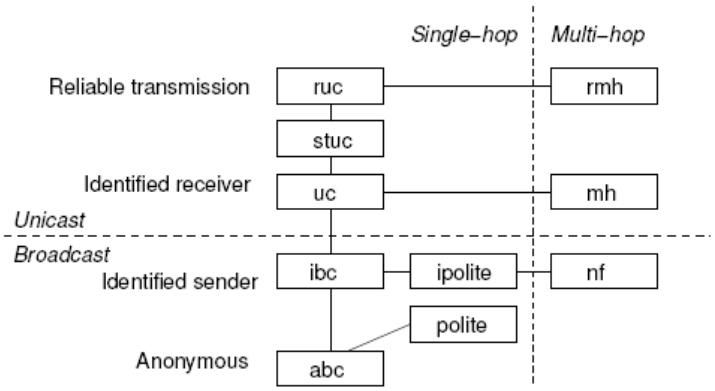

Figure 5 The communication primitives in the Rime

Layers of Rime communication stack are as follows:

- *Anonymous Best-effort Single-hop Broadcast (abc)*. The most basic communication primitive in Rime that all other Rime primitives are based on it. Using *abc* a data packet without any information about the sender is broadcasted to all local neighbors that listen to the channel on which the packet is sent.
- *Identified Best-effort Single-hop Broadcast (ibc)*. By using ibc, the single-hop sender address is added to the outgoing packets and the packets are broadcasted to all local neighbors.
- *Best-effort Single-hop Unicast (uc).* UC primitive allows unicasting a packet to an identified single-hop neighbor by adding the receiver address to the outgoing packets.
- *Stubborn Single-hop Unicast (stuc).* This primitive repeatedly (re)sends a packet to a single-hop neighbor using the uc primitive, until an upper layer primitive or protocol cancels the transmission.
- *Reliable Single-hop Unicast (ruc).* This primitive sends a packet to a single-hop neighbor. Reliability is achieved by using acknowledgements and retransmissions to ensure successful reception of the packet.
- *Polite Single-hop Broadcast (polite).* This primitive is a generalization of the polite gossip algorithm which is designed to reduce the total amount of packet transmissions by avoiding broadcast of multiple copies of a message that other nodes have already sent during a time interval.
- *Identified Polite Single-hop Broadcast (ipolite).* Works in the same way as the polite primitive, but adds the identity of the sender as a packet by using the ibc layer.
- *Best-effort Network Flooding (nf).* Floods a single packet to all nodes in the network, by using polite broadcasts at every hop to reduce the number of redundant transmissions. It sets the end-to-end sender and packet ID on outgoing packets and avoids retransmissions.

- *Best-effort Multi-hop Unicast (mh).* This primitive unicasts a packet to an identified node in the network by using multihop forwarding. The routing function for selecting the next-hop neighbor is supplied by applications.
- Hop-by-hop Reliable Multi-hop Unicast (rmh). Works similar to the *mh* primitive except that it uses the *ruc* for communicating between two single-hop neighbors.

The Rime stack supports both single-hop and multi-hop communication primitives. Abc is the lowest level in Rime which provides a 16-bit channel without addressing. Addressing and other features are added by upper layers via adding header fields for desired functionality of the implemented protocol. Detailed description of each layer can be found in [7].

Communications using Rime utilize different logical channels. Each channel has its own set of protocols and attributes. These logical channels are opened at run-time and the communicating parties must agree in advance on the particular set of protocols to be used for a particular channel. For instance, two applications running on two different nodes can communicate with each other using two logical channels one for multi-hop unicasts and the other for anonymous broadcasts.

For sensor networks, the lightweight layering principle has several benefits. Since the communication primitives are simple, they are easy to implement and test. The memory footprint of the implementations of the primitives is small, which is important for memory-constrained sensor nodes. As applications may attach to any layer of the stack, the applications can express precisely how much of the communication features that they need. Therefore, in this thesis we use Rime for the implementation of clock synchronization algorithms.

2-2-2-2 System Clock

In Contiki, the logical clock is a counter variable that counts the number of timer interrupts. The Contiki has support for 3 timers: timer, event timer (etimer) and real-time timer (rtimer).

The Contiki kernel does not provide support for timed events [6], so applications should explicitly use setting, resetting, restarting and checking expiration functions in the timer library. Applications that need to receive an event when a timer expires cannot use the timer library functions and should use event timers instead. When the event timer expires, an event will be

posted to the process that set the timer. The real time timer is actually used to handle the scheduling and execution of real-time tasks.

The code for real-time timer is architecture dependant. In MSB-430 nodes, the source of real-time clock can be selected to be either an external 32.768 kHz crystal oscillator, or processor cycles executed by the CPU (2.4576 MHz). MSP430 Basic Clock Module is shown in Figure 6.

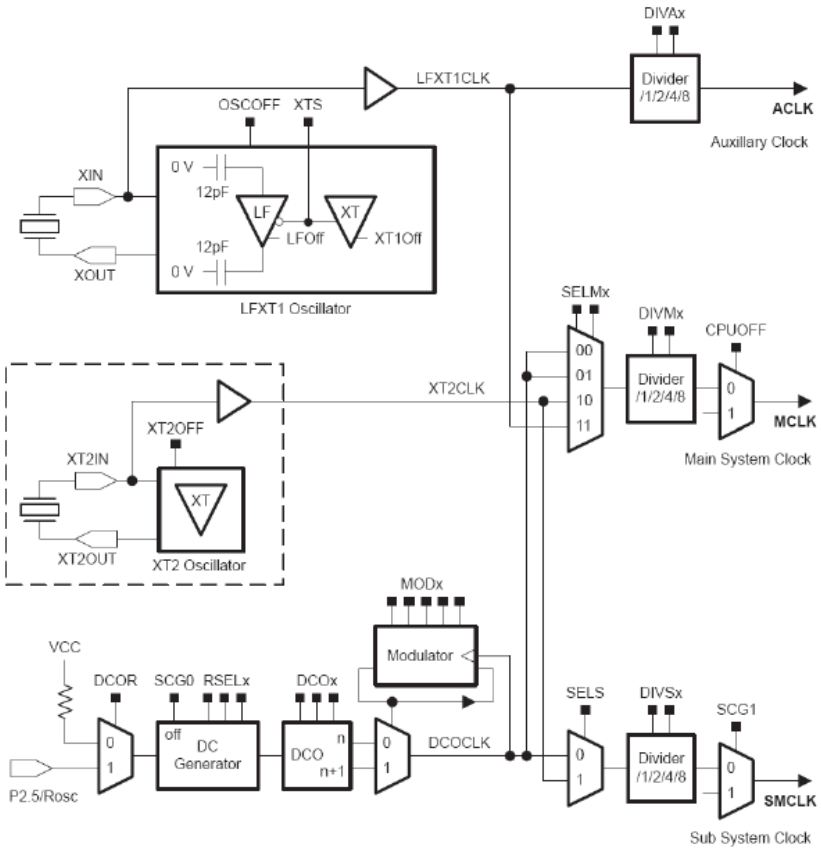

Figure 6 MSP430 Basic Clock Module

By selecting the auxiliary clock (ACLK) and 1,2,4 or 8 as a divider, the real-time timer will count the 32.768 kHz external crystal ticks. By default the Contiki code of the MSP430 rtimer selects ACLK with divider of 8 that gives clock resolution of around 244 microseconds.

By selecting the subsystem clock (SMCLK) and 1, 2, 4 or 8 as the divider, the real-time timer of Contiki will count the processor cycles executed by the 2.4576 MHz CPU. This clock can give resolution of around 3.25 microseconds when selecting 8 as the divider.

Depending on the application that needs clock synchronization and required clock granularity, any of these clocks can be used as the local clock of the nodes. Unfortunately the SMCLK clock wraps so fast that the overhead to

make sure that each wrapping is caught could adversely affect the rest of the system. Another problem is that SMCLK is disabled when the CPU is in sleep mode, so this could make disastrous results. Since the clock behavior is not deterministic and the clock value is not monotonically increasing, calculation of offsets and skews using methods such as linear regression can return erroneous values. Therefore, we use the 32.768 kHz external crystal for clock synchronization.

2-2-2-3 MAC Layer and Radio Driver

The Contiki codes for MSB-430 platform currently only provide support for a so called "NULLMAC" as the MAC layer protocol. A NULLMAC is a MAC protocol implementation that does not do anything NULLMAC, but is equivalent to the IEEE 802.15.4 specification for non-beacon based transmissions. The code for NULLMAC, on one hand, reads the data ready to be sent from Rime buffer and sends it to the radio driver, and on the other hand, gets the received packets from radio driver, clears the Rime buffer and writes the received data to the Rime buffer.

MSB-430 nodes use Chipcon CC1020 radio transceiver which is a single chip low power RF transceiver for narrowband systems with 8.6dBm maximum transmission power. CC1020 is a single-chip UHF transceiver designed for very low power and voltage wireless applications. It has a frequency range of 402 MHz – 470 MHz and 804 MHz – 940 MHz, which can be selected independently for receiver and transmitter by software, letting usage of multiple radio channels for advanced routings. Also the voltage supply is very low in range of 2.3V to 3.6V.

The CC1020 code in the Contiki operating system configures the radio transceiver and receiver, gets the data to be sent from the MAC layer and creates the packet by adding preamble and Synchword before the data and a tail after it. Radio driver waits for the medium to become idle (carrier sence) and then waits for a short pseudo-random time before sending (0-1500μs). Finally it switches to transceiver mode and initiates the radio transfer.

When the radio is in receiver mode, arrival of a packet leads to an interrupt. The interrupt handler gets data byte by byte and puts it in the CC1020 receive buffer. When reception is complete, radio driver copies the data from CC1020 rxbuffer to the MAC layer.

Some of clock synchronization algorithms, such as TSPN and FTSP, need that the time-stamping to be done at MAC layer, so having the knowledge about MAC layer and radio driver is essential for implementation of such algorithms.

CHAPTER 3

DESIGN AND IMPLEMENTATION

This chapter describes requirements, models, and design and implementation-related details of the selected clock synchronization algorithms.

3-1 SYSTEM MODEL

The system consists of a number of nodes distributed in arbitrary locations in a test environment. Depending on the application we can assume either fixed or random placement of the nodes. Each node in the network has a unique identifier which is burnt into its memory. We have employed MSB-430 nodes and used the existing Contiki port on this platform. The algorithms are implemented using the C programming language. MSB-430 nodes are modular and different types of sensors can be added to it. We decided to use the Rime communication stack primitives to broadcast synchronization messages. The communication range of nodes depends on the antenna that can optionally be attached to the nodes. In wireless communications broadcast messages may collide if nodes are in each other's communication radius or due to the hidden terminal effect. Synchronization messages may also get corrupted because of media noise generated by cellular phones, wireless access points and so on. These faults should be considered when trying to implement a robust and secure clock synchronization algorithm.

If we want to deploy sensor nodes outside the research environment, we should take into account the environmental factors that might affect the system such as wind, rain, heat, battery level and so on.

The most important modules required for implementing a fine-grained synchronization protocol is the system clock. For synchronization we used the

external 32.768 kHz quartz which is accessed by calling architecture dependent rtimer libraries of Contiki. The Rtimer module handles the scheduling and execution of real-time tasks. The real-time clock is a 16-bit counter which counts the ticks generated by the external crystal. This clock is bounded and resets when it reaches 65535. We have implemented a 32-bit local clock which uses a 16-bit counter that is increased each time the hardware interrupt occurs and the 16-bit rtimer wraps around.

3-2 IMPLEMENTATION OF RBS

The fundamental property of Reference-Broadcast Synchronization [3] is that a broadcast message is used for synchronizing a set of receivers with one another, not with a sender. In RBS, a reference message is sent without any time-stamp or timing information using the radio broadcast. A message that is broadcast at the physical layer will arrive at a set of receivers with very little variability in its delay, so the receivers can use the arrival time of the message as a point of reference for comparing the global time.

In a single broadcast domain (single hop network), the RBS algorithm is as follows. A transmitter broadcasts *m* reference packets. Each of the *n* receivers records the time that the reference was observed, according to its local clock. Then the receivers exchange their observations. Each receiver *i* can compute its phase offset to any other receiver *j*, and can correct its clock skew by using least square linear regression.

In order to have a completely distributed synchronization we suggest that all of the nodes in the network participate in transmission of beacon messages. In many implementations a special node is used for broadcasting beacons and other nodes just transmit exchange messages. But in our implementation we decided that nodes take turn to broadcast beacon messages (see chapter 5 for more descriptions).

In our implementation on Contiki O.S. for MSB-430 motes, the synchronization process on each node periodically checks an event timer (etimer) in an infinite loop (while(1)). Upon expiration of this etimer, node checks whether it is the round for it to broadcast a beacon or not according to its unique node ID. We also used a round counter variable to count the number of etimer expirations, if this counter divided by number of nodes in the cluster is equal to the node ID, the node starts creating a beacon packet. A cluster is a one-hop network in which all nodes are in communication range of each other and can send and receive messages to/from each other.

For broadcasts we have defined two logical channels, one for broadcasting reference messages (beacon) and one for sending exchange messages that

contain the reception time of the reference broadcast. The beacon packet contains a beacon counter number which is increased for each packet. After creation of the beacon it is copied to the Rime buffer and is broadcasted.

Upon receipt of a beacon message, receiver nodes read their local time which is the value of the native clock at that moment (by calling the RTIMER_NOW() function) together with the value of the counter that counts the number of clock wrap arounds (rtimer_counter). The rtimer_counter is incremented in an interrupt handler function which is executed each time the clock value changes from 65535 to zero. The time-stamping can be done either in the synchronization code on top of Rime, or in the interrupt handler of CC1020 radio driver. Since by time-stamping at lower levels synchronization error can be reduced, we read the clock in the interrupt handler of the radio driver for our implementation.

This recorded local time is copied to a buffer together with the beacon sender's node ID and the beacon counter number. This buffer will later be placed in the exchange message and is used in the calculation of clock offset. For creation of exchange message, the node puts its own node ID, the beacon sender's node ID and the beacon counter number with the recorded arrival time in the Rime buffer and broadcasts it. This message could be sent to selected neighbors, but since we want our implementation to be completely distributed, the packet is broadcasted to all neighbors in a cluster.

As the exchange message is received by a node other than the beacon sender, the node checks whether it has received the same beacon from the same sender with the same beacon number and has stored the arrival time. The node subtracts the time in the exchange message from its recorded time to calculate the clock offset. This value is added to a table (regression table) to be used for calculation of clock skew and phase offset using least square linear regression. Each node keeps a table for each of its neighbors to be able to estimate their local time.

For least square linear regression calculation, we used the following formulas[2]:

$$Clock\ Skew: m = \frac{S_{xy}}{S_{xx}}$$

$$Phase\ \text{Offset: } (mean\ of\ y) - (mean\ of\ x) \times m$$

Where x is the value of a counter and y is a offset value and,

$$S_{xx} = \sum x^2 - \frac{(\sum x)^2}{n}$$

[2] http://pirate.shu.edu/~wachsmut/Teaching/MATH1101/Relations/linear-regression.html

$$S_{xy} = \sum xy - \frac{(\sum x)(\sum y)}{n}$$

$$mean\ of\ x = \frac{\sum x}{n}$$

$$mean\ of\ y = \frac{\sum y}{n}$$

These transmissions and calculations are never stopped in a continuous implementation to guarantee fine-grained clock synchronization.

3-3 IMPLEMENTATION OF FTSP

In Flooding Time Synchronization Algorithm [4] implementation for a single hop scenario (that all nodes can send and receive broadcast messages to each other), one node maintains an estimated global time. This node should periodically broadcast a synchronization packet containing its clock value. The time-stamp in the synchronization message will be used by all receivers to estimating the global time.

To achieve high precision, FTSP utilizes MAC-layer time-stamping to eliminate many sources of delays and uncertainties in message transmission. The FTSP records multiple time stamps both on the sender and receiver sides to reduce the jitter of the interrupt handling and encoding/decoding times.

For implementation, one of the nodes is selected to act as the synchronizer to broadcast its time as the global time of the system. This node waits inside a forever loop for expiration of an event timer which occurs every T seconds. Upon etimer expiration the node copies its node ID to the Rime buffer. The MAC layer of the communication stack, copies the packet from Rime buffer and puts it in the radio driver buffer where time-stamping happens there.

In our implementation, time-stamping has been done in the lowest possible level of the communication stack. In the sender side, multiple time-stamping is done while the packet is being transmitted to the radio and just before sending the time-stamp field of the packet.

These operations are done as part of the CC1020 radio driver implementation for the MSB-430 platform. The format of the packets being created by this driver is shown in Figure 7. Since the packet is transmitted one byte at a time, multiple time-stamps can be recorded while transmitting each byte after the SYNC bytes as they are transmitted or received. These time stamps should be normalized by subtracting a multiple of the time it takes to transmit a byte. According to [4], the jitter of interrupt handling time can be eliminated with high probability by taking the minimum of the normalized time stamps. The

jitter of encoding and decoding time can be reduced by taking the average of these interrupt error corrected normalized time stamps. Only the final error-corrected time stamp will be placed in the synchronization message before the tail field. It has been shown in [4] that by using only 6 time stamps in Mica2 platform, the time-stamping precision was improved from tens of microseconds to a few microseconds.

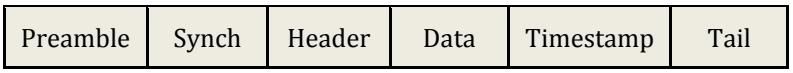

Figure 7 Data Packet

On the receiver side the packet data field and the time-stamp field are copied to the Rime buffer where the synchronization process can read from it.

Upon arrival of each byte of packet data field, a time-stamp is recorded in the interrupt handler and the recorded time stamps are averaged. The final averaged time stamp should also be corrected by the byte alignment time that can be obtained from transmission speed and the bit offset. Finally, this value is passed to the synchronization process where nodes subtract the timestamp in the message from the recorded arrival time to calculate their offset with the synchronizer. This offset value is added to the regression table and the skew and phase offset are calculated.

FTSP performs a least-squares linear regression to compensate for clock skews. This method offers a fast, closed-form method for finding the best fit line through the phase error observations over time and was first used in RBS (see 3-2). The linear regression is usually performed off-line to calculate absolute error values produced by each protocol. This method should also be implemented on nodes but due to the memory constraints of the nodes, only a limited number of data points can be stored and used. In [4] linear regression is calculated on 8 most recently calculated offset values.

CHAPTER 4

EVALUATION

To evaluate and compare our implementation with those mentioned in the original papers, a series of experiments were conducted using MSB-430 platform.

4-1 RBS

The RBS [3] algorithm was originally implemented on Berkeley Motes with clock resolution of 2μs, and on StrongARM-based Compaq IPAQs with 1μs clock resolution. In the first configuration, 5 Berkeley Motes were periodically broadcasting a reference pulse with a sequence number. Each of them time-stamped the reception times of incoming broadcasts. Then an offline analysis of the data was performed. In their experiment, the residual error was 11.2μs.

We implemented a similar scenario on MSB-430 motes, where 4 motes were periodically sending reference broadcast. Each mote was time-stamping receipt of the broadcast with a 32768Hz clock which gives us a resolution of 244μs. The maximum absolute error was 3.7 ticks (903μs) and the average value was 1.3 (317μs).

Figure 8 depicts the phase offset of the clocks of two nodes after receiving reference broadcasts. The best fit line has been calculated offline. The slope of the best-fit line defines the clock skew (-0.01757 here), and the line intercept defines the initial phase offset (25.82785 here).

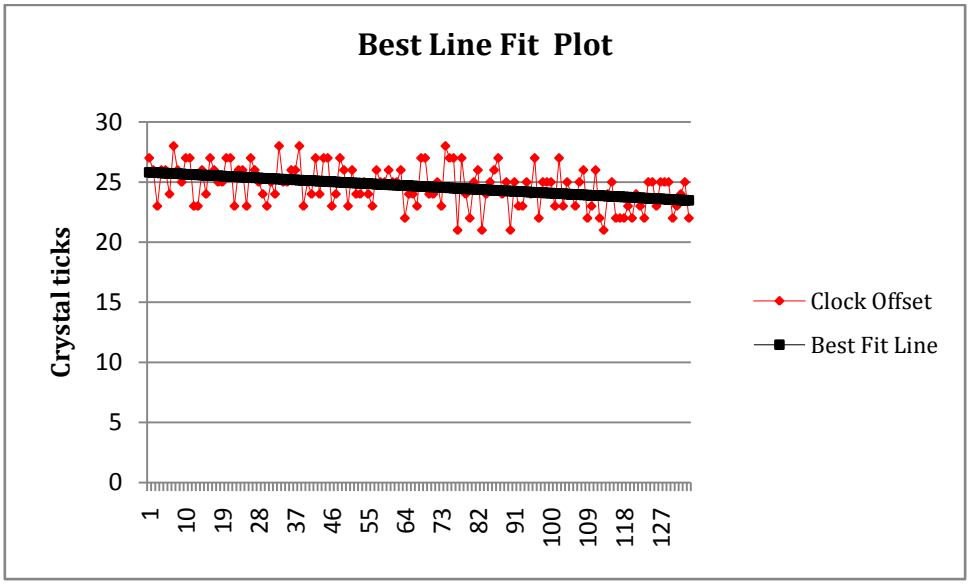

Figure 8 Analysis of mote's phase offset and a least-square-error fit to them

The synchronization error can be reduced if the clock can be read at interrupt time in the interrupt handler, before protocol processing [3]. Figure 9 illustrates the error between the real global time and estimated global time when time-stamping beacon arrival in the interrupt handler of the CC1020 radio driver.

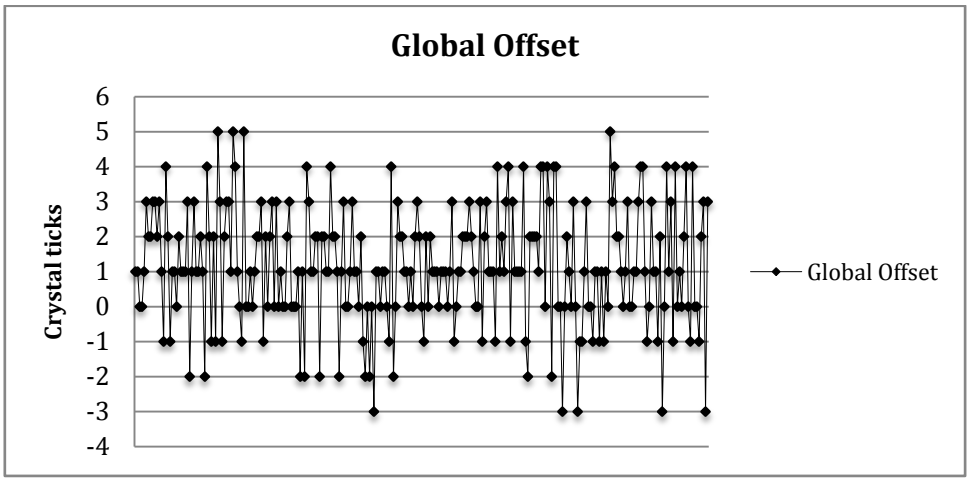

Figure 9 Phase offsets of estimated global time and the real global time

4-2 FTSP

The FTSP [4] algorithm was originally implemented on Mica/Mica2 platforms running the TinyOS operating system. Mica2 motes use a 7.37 MHz clock which has a resolution of around 1μs.

The following experiment was used to show that with only 6 time stamps, the time-stamping precision can be improved from tens of microseconds to 1.4μs on the Mica2 platform. 4 motes send time-stamped messages to each other for 10 minutes, each with a 5-second sending period. The timestamps should be recorded both on the sender and receiver sides, and the pair-wise clock offset and skew values are determined offline with linear regression. The time-stamping error is the absolute value of the difference of the recorded receiver side time stamp and the linearly corrected sender side time-stamp. The average and maximum time-stamping errors on Mica2 platform were 1.4μs and 4.2μs, respectively.

In MSB-430 platform running Contiki using a 32.768 KHz clock, the maximum and average time-stamping errors were 2.01 ticks (491μs) and 0.84 ticks (205μs), respectively. Figure 10 shows the distribution of absolute errors for 360 collected data points. As it can be seen, 62% of the collected data points had a time-stamping error between 0 and 1 clock tick (0-244μs), and only 1% of the collected data had error equal or higher than 2 ticks.

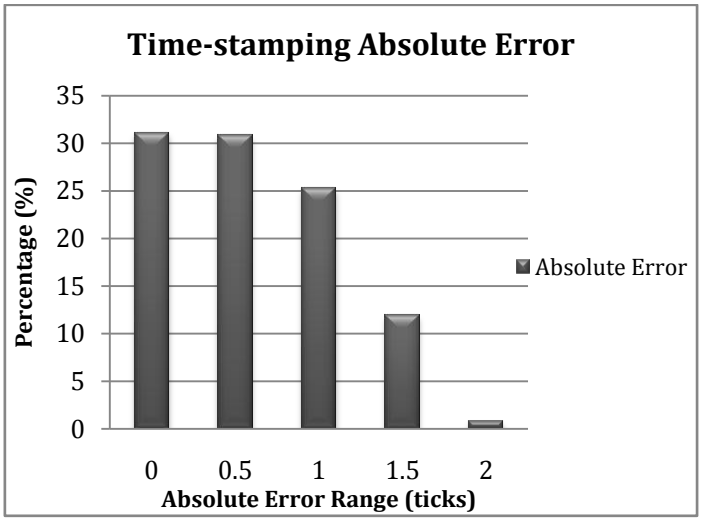

Figure 10 Distribution of Absolute Error Values

The offset between the two clocks changes in a linear fashion provided that the short term stability of the clocks is good [4]. The stability of the 7.37 MHz

Mica2 clock was verified in [4] by periodically sending a reference broadcast message that was received by two different motes. The two motes time-stamped the reference message using the FTSP time-stamping described in the previous section with their local time of arrival and reported the time-stamp. For each transmitted message the offset of the two reported time-stamps was calculated. The offsets were further examined by linear regression. A one hour experiment produced the following results on Mica2 platform, the average value of the absolute errors was 0.95μs and the maximum absolute error was 4.32μs.

For the MSB-430 motes, the stability of the 32768 Hz clock was verified with the same scenario. The average value of the absolute errors was 1.13 ticks (276μs) and the maximum absolute error was 3.23 ticks (780μs).

In order to identify the trend of the global time relative to the local time from the data points received in the past, the following scenario was used to test Mica2 implementation in [4]. Mote A which maintains the global time sends synchronization messages to mote B with a period of *T*. Mote B estimates the skew and offset of its local clock from that of A using linear regression on the past 8 data points. A reference broadcaster sends a query message with period *t* and both A and B respond to this query by time-stamping its arrival with the global time and reporting it to the base station. For *T = 30* seconds and *t = 18* seconds, the average absolute error was 1.48μs, and the maximum absolute error was 6.48μs.

In case of MSB-430 implementation, the same experiment resulted to the average absolute error of 3.9 ticks (967μs), and the maximum absolute error of 9 ticks (2197μs). We also repeated this experiment with *T = 0.4* seconds and *t = 0.2* seconds to verify that by reducing the period of sending synchronization message, how much the precision of clock synchronization can improve. The average and maximum values changed to 3 and 8 ticks respectively. So by decreasing the resynchronization interval from 30s to 0.4s, the results only improved slightly. This improvement comes with a cost of sending more messages in each round which consumes around 75 times more energy, while only improving the precision around 1.12 times.

Figure 11 illustrates the result of applying clock synchronization on mote's local clock. It is clear that before applying clock synchronization the estimated global time is actually the time which is read from the local clock of the node. When enough synchronization messages are received, each mote can estimate its clock skew and offset with the global time and compensates for them.

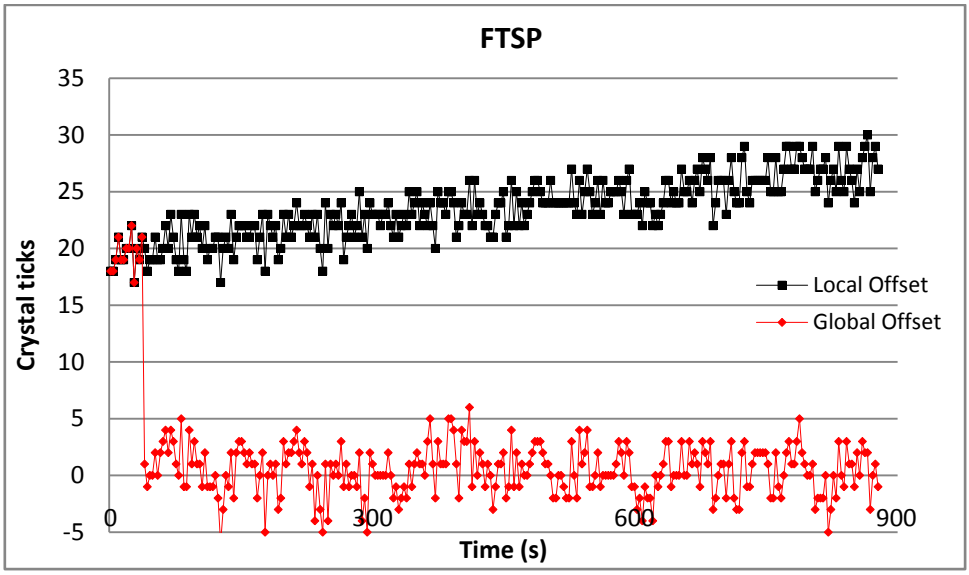

Figure 11 Comparison of synchronized and local times for MSB-430 motes, T = 3s and t = 1.8s

Figure 12 shows the error between the real global time and the estimated global time after clock synchronization starts, for both system clocks.

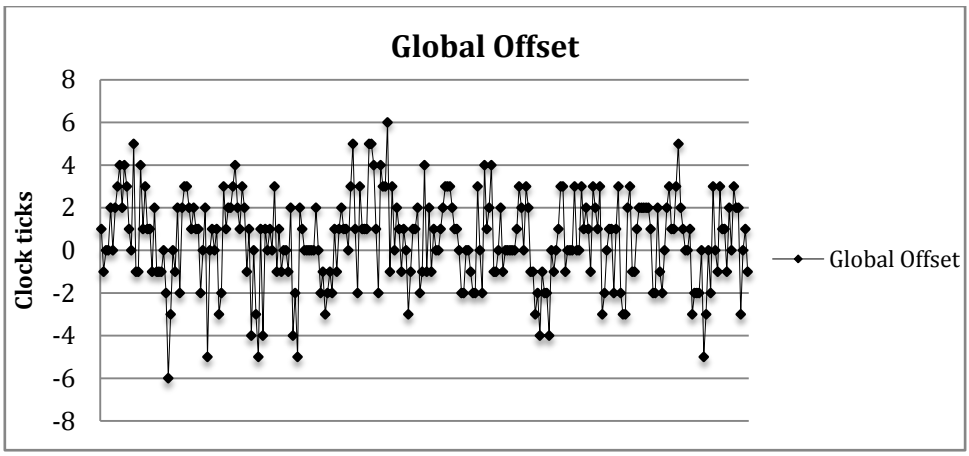

Figure 12 Phase offsets of estimated global time and the real global time for T=0.4s and t=0.2s

4-3 COMPARING FTSP AND RBS

To be able to compare FTSP and RBS algorithms for clock synchronization, we need to conduct identical experiments for both implementations. We decided

to apply a scenario similar to the last experiment of FTSP that can be applied to RBS. Mote A sends a reference broadcast with period *T*, two other motes, B and C, time-stamp the arrival of the beacon with their corresponding local clocks. A fourth node sends a query broadcast with period *t*. Both B and C time-stamp arrival of query message with their estimated global time and report it. Figure 13 demonstrates the difference between the offset of local time and global time on one of the nodes.

The maximum and average error values computed in this scenario can be compared with those of FTSP with equal *T* and *t* periods.

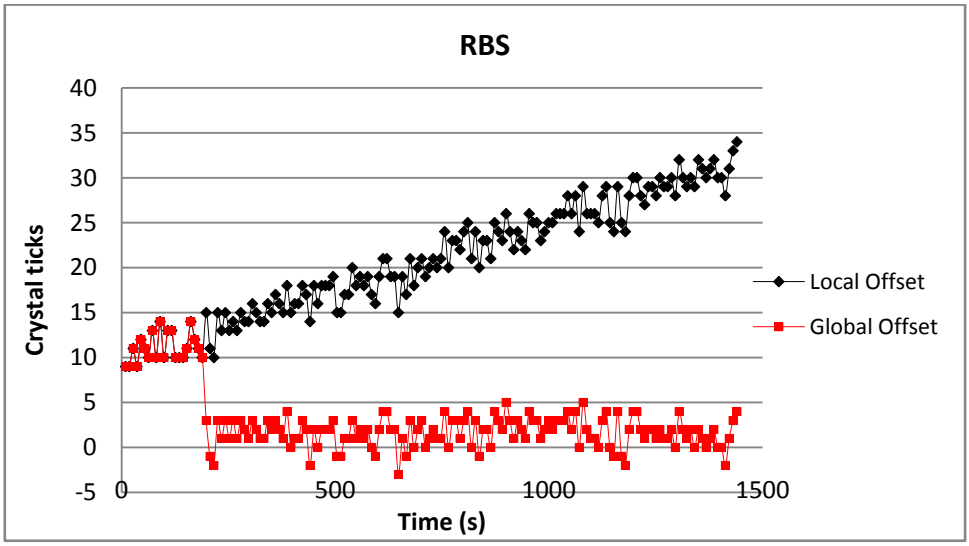

Figure 13 Comparison of global and local times for MSB-430 motes

For the MSB-430 motes with the 32768 Hz clock and T = 3s and t=1.8s, the maximum absolute error value was equal to 9 crystal ticks (2197μs), and the absolute average error was around 1.517 ticks (370μs).

4-3-1 Absolute Synchronization Error

In Figure 14 the distribution of absolute values of clock differences among 4 sensor nodes using RBS for their synchronization, a synchronizer sensor node and other nodes running FTSP are shown. A fifth node was used to send a query message, and every node in the network reported its global time upon arrival of this query message. It can be seen that when employing RBS, at around 17% of reported times, the nodes were accurately synchronized and reported exactly the same global time. But for FTSP, only for around 7% of the

queries, nodes reported the same global time with no synchronization errors. Using RBS, in 80% of queries, the global times reported by nodes were 1 to 4 ticks away from each other, and only 3% of clocks had 5 to 8 ticks difference with each other. However, when FTSP was in use, synchronization error for almost 25% of queries was in range of 5 to 8 clock ticks.

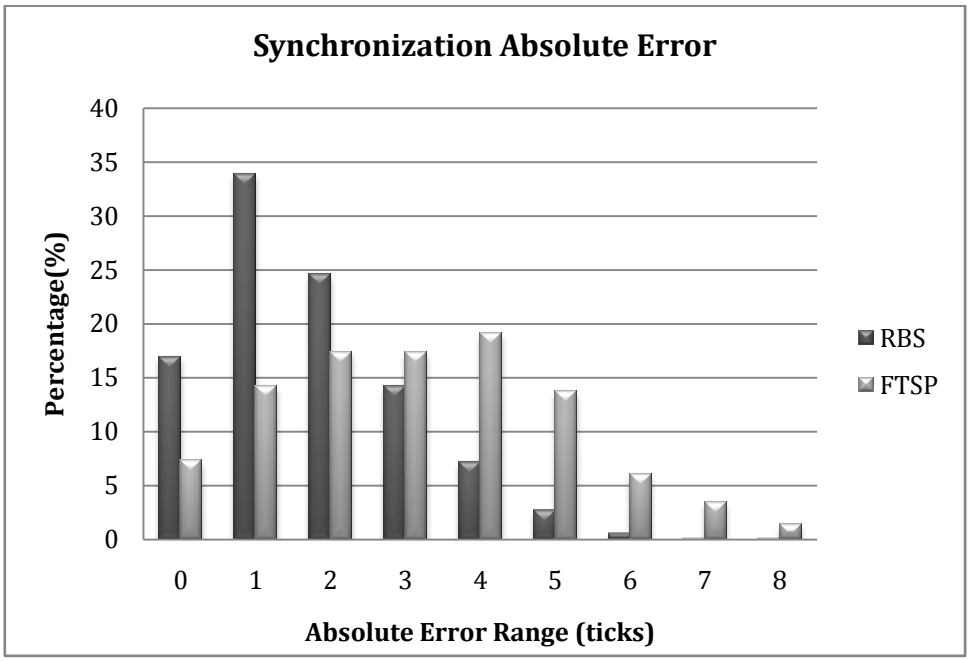

Figure 14 Absolute error of nodes' clocks with one another using RBS v.s. FTSP

Although Maróti etc. [4] claimed that FTSP achieves more precise synchronization and fewer errors than RBS, but in our implementation on MSB-430 motes using Contiki operating system, they both had the same absolute maximum error value but RBS produced less absolute average error.

One reason for this result is that although the FTSP approach is really simple and efficient, but its performance is completely dependent to the operation of the radio driver. Therefore, depending on the platform on which it is being implemented the precision can vary. For instance in many sensor nodes it is not efficient to transmit data one byte at a time.

4-3-2 MESSAGE COMPLEXITY

Communication or message complexity is used for measuring the traffic load of the system. This is achieved by counting the total number of messages that

are exchanged among the nodes in the worst case. For calculating the communication complexity, the size of messages also might be of interest.

As described in [4], if the resynchronization period is T seconds, then each node sends 1 message per T seconds in FTSP, and 1.5 message per T seconds in RBS (0.5 for a reference broadcast and 1 for a time stamp exchange message). So in each round using FTSP, only 1 message with a constant size is transmitted in a single-hop network, but in RBS, on the other hand, 1 beacon and $n-1$ exchange messages are being transferred in the network, where n is the number of nodes in a cluster.

4-3-3 Energy Efficiency

For FTSP approach, all the sensor nodes in the network except the synchronizer are always in listening mode. Every T seconds which is the pre-defined synchronization period, the synchronizer broadcasts the synchronization message. So the synchronizer can stay in low power mode state and changes state to transmission state every T seconds conserving more energy. When the nodes' clocks become synchronized they can also wake up every T seconds to receive the synchronization message.

The only problem with FTSP is that since we have to transmit the synchronization message byte-by-byte to perform the time-stamping, the transmission takes more energy. So if the synchronizer is same as the other nodes in terms of its source of energy, its battery will run off much faster than other nodes in the network.

The simplest implementation of RBS requires that a reference node broadcasts a beacon every T seconds. The rest of the nodes should stay in listening mode to receive the packet, then the nodes change state to transmission state and transmit an exchange message as the receive response. After transmission the nodes go back to listening state to receive exchange messages from other neighbors. However, in our implementation of RBS, all nodes in the network participate in sending and receiving synchronization messages. In each synchronization round (every T seconds) one of the nodes broadcast a reference message.

It can be seen that in FTSP, nodes consume less energy for performing the synchronization process, because, in every synchronization round, FTSP requires one message transmission to be done by the synchronizer node, and reception of one message by other nodes. However, for simple implementation of RBS, reference node broadcasts one message and any other node requires to receive $1 + (p-1)$ messages and transmits one exchange message in every round, where p is the number of nodes in the network that

each node should receive its exchange message. In [3], *p* is considered to be equal to one, meaning that each node only needs to send and receive exchange messages from one neighbor.

In our implementation of RBS, since there is no need for a particular reference node, each node transmits a beacon packet every *n* synchronization rounds (where n is the number of nodes in the single-hop network). Therefore, all the nodes consume same amount of energy and only transmit one extra message every *n* rounds.

4-3-4 SPACE COMPLEXITY

Since the sensor nodes are typically very constrained in their memory, it is important to take the total number of memory bits used to implement the algorithm into account.

For implementation of FTSP we had to keep a table (regression table) and some other variables such as rtimer_counter. In the regression table, nodes keep 8 of the most recently calculated offset values together with a value that shows the number of received synchronization messages. Nodes also need to temporary keep multiple time-stamps at the radio driver to perform averaging and error reductions.

In case of basic RBS algorithm, each node has to keep a regression table and some variables, similar to the FTSP. The nodes should also keep a buffer containing latest received beacon number and its arrival time. Therefore, the memory required for implementation of both approaches is more or less the same.

But if we implement RBS in a way that all nodes participate in beacon transmission and keep track of offset and skew with all other neighbors, the space complexity of this algorithm is dramatically increased. This is because nodes should keep offset values with *n-1* nodes in their regression table, where *n* is the number of nodes in a neighborhood. They should also keep a variable containing the beacon number to be used in their broadcasts, and the node ID of the last received beacon to be used in exchange messages.

Another important criterion that can be taken into account for comparing clock synchronization algorithms is their robustness to node and link failures which is described in more details in the next chapter.

CHAPTER 5

FAULT TOLERANCE

One of the most important requirements of a clock synchronization protocol for sensor networks is that it should be robust to failures and antagonistic. We want our sensor network to stay synchronized even if some nodes fail, new nodes join, adversaries tamper with the network, and messages get lost due to collisions or noise.

In this chapter we review some of the faults and failures that can happen during a sensor network lifetime and the next chapter will cover some of the security issues that can affect the correctness of clock synchronization protocols. Unfortunately, due to the unattended environment where sensor networks typically reside, it is not possible to consider all the faults and attacks that can happen and compensate for them, and sometimes it is even difficult to detect a fault or combination of faults from an adversary attack.

5-1 FAULT MODEL

Wireless sensor networks are susceptible to a wide variety of faults due to undetermined environmental conditions, hardware limitations and software bugs. In this section we briefly mention some of the possible faults and failures that can happen in a sensor network lifetime.

During deployment sensor nodes may be dropped from height and break down. In some cases sensors may get damaged due to heat or moisture in the harsh environment and do not survive. The battery of sensor nodes can run out and the node stops working. In an unattended environment adversaries can destroy or still the nodes to cause producing erroneous output by the sensor network.

Message omission failures happen more often in wireless sensor networks than traditional wired networks. Message transmission can lead to message losses due to collisions when two or more nodes in communication range of each other transmit simultaneously, or due to the hidden terminal effect. Even in absence of collisions, message still might get lost because of random media noise or fading during propagation over the wireless medium.

Limited computational resources on sensor nodes can impose some limitations on the amount of processing that nodes can perform successfully. If this limit is exceeded, processing tasks may cause non-deterministic behavior and different kinds of failures [27]. Buffers may overflow, memory locations might be overwritten, pointers and memory locations can corrupt, and important events can get lost. Certain conditions may lead the nodes into deadlock states or continuous restarts by watchdog timers.

In the following subsections we describe the effect of some of the aforementioned faults and failures on the operation of synchronization algorithms.

5-2 FTSP

In this section we review some of the most important effects of node and communication failures to FTSP [4] approach and describe the solutions we used in our implementations.

5-2-1 NODE FAILURE

In FTSP, one of the nodes keeps track of the global clock and broadcasts a synchronization message containing a timestamp, allowing others to synchronize their local clocks to it. In this method the synchronizer node is a single point of failure, if it fails the other nodes have no reference to get synchronized to, so their clocks will eventually drift away.

One solution to this problem is implementing a sort of leader election algorithm. Leader election is the process of designating a single node as the synchronizer of the network after start-up and in case of leader failure. A distributed leader election algorithm is described in the following subsection section.

5-2-1-1 LEADER ELECTION

As mentioned before, FTSP is a centralized synchronization algorithm in which the synchronizer node becomes a single point of failure. To overcome this weakness, a robust leader election algorithm is required to select a leader at start-up and after the primary leader's failure. The algorithm should be distributed and must guarantee that only one node at a time in a network will be elected as the leader.

In [4] a root election algorithm is proposed that selects a node with smallest node ID as the root for multi-hop synchronization. But in order to achieve fault tolerance, even in a single-hop network leader election is necessary and should be implemented.

The algorithm proposed in [4] works as follows:

Each node in the network has a unique node ID (myNodeID) and maintains a variable containing the ID of the leader (root) of the network (myRootID). Each synchronization message has fields containing root ID which contains the ID of the sender and sequence number which is increased by one for each new message. Nodes also maintain a variable to keep track of the sequence number of the synchronization messages (highestSeqNum). These variables are updated upon arrival of a new synchronization message. If a node does not receive the broadcast for a ROOT_TIMEOUT period, it declares itself as the new leader (myRootID = myID) and broadcasts a synchronization message containing the global time that it has computed using its skew and offset with the first leader. Whenever this node receives a message with a node ID smaller than its own, it updates its myRootID variable and stops broadcasting the global time. This mechanism guarantees that nodes with higher IDs will give up and eventually only a node with smallest node ID will become the leader in the network. The pseudo code of the leader election algorithm as described in [4] is shown in figures 15 and 16.

To avoid inconsistencies, it is proposed in [4] that only root and those nodes that have enough entries (NUM_ENTRIES_LIMIT) in their regression table are allowed to broadcast synchronization messages. Nodes receiving a new synchronization message that disagrees with values stored in their regression table should clear the table and start gathering reference points. More detail about this algorithm can be found in [4].

```
1    Event Receive (TimeSyncMsg * msg)
2    {
3      If ( msg->rootID < myRootID )
4        myRootID = msg->rootID;
5      else if ( msg->rootID > myRootID || msg->seqNum <= highestSeqNum )
6        return;
7
8      highestSeqNum = msg->seqNum;
9      if (myRootID < myNodeID)
10       heartbeats = 0;
11
12     if ( numEntries >= NUMENTRIES_LIMIT
13         && getError(msg) > TIME_ERROR_LIMIT)
14       clearRegressionTable();
15     else {
16       addEntrytoRegressionTable();
17       calculateLinearRegression();
18     }
19   }
```

Figure 15 Handling of a new synchronization message

```
1    Event Timer_Expire()
2    {
3      ++ heartbeats;
4      if (myRootID != myNodeID && heartbeats >= ROOT_TIMEOUT )
5        myRootID = myNodeID;
6
7      if ( numEntries >= NUM_ENTRIES_LIMIT || myRootID == myNodeID ) {
8        msg.rootID = myNodeID;
9        msg.seqNum = highestSeqNum;
10       Broadcast(msg);
11
12       if ( myRootID == myNodeID )
13         ++ highestSeqNum;
14     }
15   }
```

Figure 16 Periodic broadcast of a synchronization message

In our implementations we defined the ROOT_TIMEOUT value to be 5 seconds and NUM_ENTRIES_LIMIT to be 8 entries. According to the sensor network application and environmental conditions in which sensor nodes are deployed these value can be adjusted. For example, in a noisy environment where message omission failures are high the ROOT_TIMEOUT value can be increased, or in case of very memory and computational power constrained sensor nodes, less number of entries can be used for linear regression calculations. Therefore, there is a trade-off between clock synchronization precision and computational resources, also between drifting away from synchronized time and the cost of performing unnecessary leader election procedure.

5-2-2 NODE (RE) JOINING

Another situation that can happen in real environment is that a new node joins the network or a node restarts due to software, hardware or environmental problems. When a node joins the network or restarts, its initial offset with other nodes will be very high. Since sensor nodes are resource constrained, computations such as calculating the average of timestamps in the radio driver, or computing least square liner regression with large offset values may lead to problems such as buffer overflows. These errors prevent the newly joint nodes to accurately get synchronized and negatively impacts synchronization of other nodes. Figure 17 illustrates an example of node restart and the global time offset calculated after trying to resynchronize.

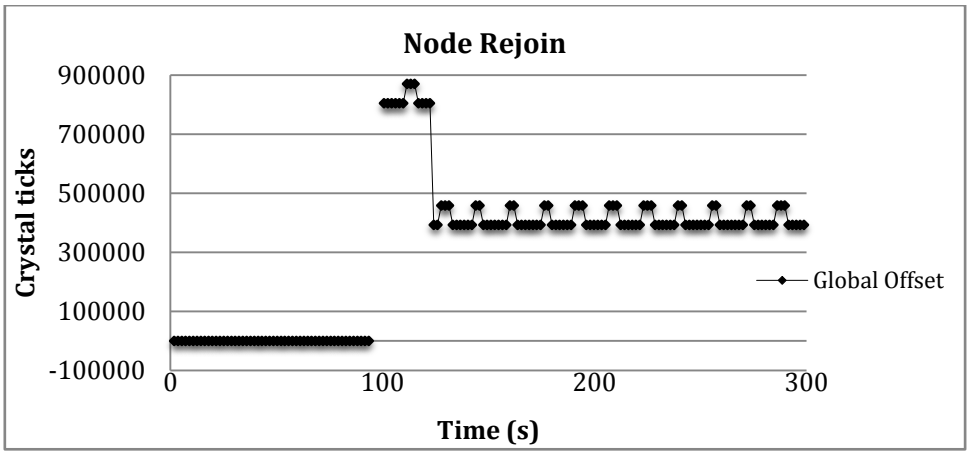

Figure 17 Effect of node restart after 100 seconds

To avoid problem that can happen by (re)joining nodes to the network we propose a simple solution.

In our implementations we have used a 2-byte value for holding the native time of the node and a 2-byte value containing a counter that counts the number of times the native clock wraps around. These two values are used for calculating the local time of each node. On start-up or restart, these values are set to zero and monotonically increase during time. By sending these values in the synchronization messages, the nodes can calculate the real global offset. We define a threshold value for the calculated offsets. If the receiver node's calculated offset with the transmitting node is more than the defined threshold, receiver figures out that a problem has happened. If the receiving node's local time is larger than the other node, meaning that the other node has a time stamp which is far in the past, the calculated offset is ignored.

On the other hand, if the receiver has a local time which is smaller than the sender, meaning that the other node has a time stamp which is far in the future, receiving node realizes that it should adopt itself to the faster node. Thus, the receiver node changes its counter value to that of the faster node and calculates the local and global time according to that. In this way, the offset between nodes will be kept bounded to [0, 65535] crystal ticks, therefore, less computation errors will occur. Figure 18 illustrates a node restart scenario using thresholds to adopt the counter value.

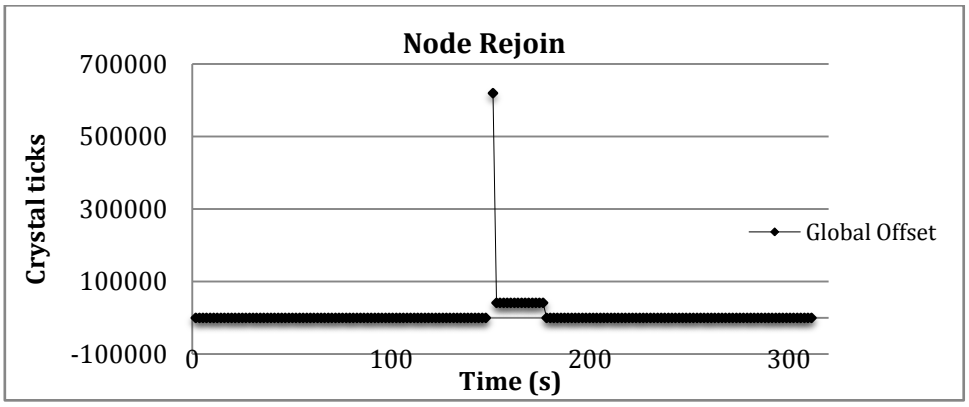

Figure 18 Effect of node restart after 150 seconds

The only limitation imposed by linear regression calculation is that we need to wait for a number of entries in the regression table before being able to calculate a precise global time. Depending on the number of required entries, the time it takes for a node to (re)synchronize can vary.

Another problem can happen when the node joining the existing network has a native time and counter value greater than that of other nodes in the network. The previously described method will result in a situation that all the present nodes in the network will try to adjust their counters to that of the new node. This will introduce synchronization errors especially in a large network. To recognize this situation we simply take advantage of the sequence number field which is present in the FTSP protocol messages.

In FTSP the synchronization message contains a field for sequence number. This simplifies the recognition of a node (re)joining to the network. If a node receives a beacon with a sequence number less than what it has previously seen from the same node, it realizes that this node has been restarted, so instead of inserting the offset to the regression table, it clears the table and waits for the node to adopt itself to the network. If a node receives a beacon from a node for the first time with a large beacon sequence number or with a time stamp far in the future, it will reject the message as an outlier.

5-2-3 COMMUNICATION FAILURE

Communication messages get lost in wireless environment due to collisions, noise, fading effects and so on. Message loss can adversely affect the synchronization procedure and lead to timing errors.

In FTSP since the only node that transmits is the synchronizer, there will be no messages collisions. The only time that collisions can happen is during the leader election process when different nodes can start broadcasting at the same time. There are different solutions for this problem. Nodes can either wait for a random time before starting transmission or they can wait for their dedicated time slot. These two approaches to collision avoidance are described thoroughly in later subsections (see 5-3-3).

Message losses due to random media noise can always happen. In FTSP loss of synchronization messages can cause the (re)synchronization procedure to take more time. As described in [4], nodes need at least 8 data points in their regression table before they can calculate their clock skew and phase offset with the synchronizer. If the number of message losses is high, it takes more time before the synchronization can start.

Another problem that can happen when message losses are high is that the nodes may not receive consecutive synchronization messages and conclude that the previous leader has failed. In this case, nodes start broadcasting synchronization messages for leader election and can cause more message collisions. Therefore, the timeout value for starting the leader election procedure should be adjusted considering the noise in the environment in which the nodes are deployed.

5-3 RBS

In this section effect of node and communication failures to RBS and some of the possible solutions are described.

5-3-1 NODE FAILURE

In RBS, if there is only one node broadcasting the reference messages, a single point of failure is introduced to the system. Many implementations of RBS algorithm assume that there is a dedicated node to act as the reference to continuously broadcast beacons. If this node fails the sensor nodes will get out of synch. Another problem with this implementation is that the reference

broadcasting node can never get synchronized so this node cannot be used as part of the network for sensing and gathering data.

But we have implemented RBS in a way that nodes take turn to broadcast beacons, therefore there will be no single point of failure in the system and all the nodes can be synchronized and collaborate in the data gathering process.

The only limitation imposed by RBS is that, there is a need that at least 3 nodes are present in the network for time synchronization algorithm to succeed (one sending the beacon and the two other nodes exchanging the beacon's arrival time according to their local clocks). This is not a big challenge since most of the sensor network applications deploy hundreds or thousands of nodes for their purpose.

Another challenge that must be taken into account is the number of neighbors that each node sends to and receive exchange message from. For simplicity and reducing the number of message transmissions, one may decide that each node only exchange messages with one other neighbor. In this way if one of these nodes fails, the other will not be able to get synchronized. Therefore, either a method for selecting a new neighbor for exchanging messages should be implemented, or more nodes exchange their timing information to add redundancy.

The most robust and reliable implementation of RBS is an implementation where any node in a single-hop network sends to and receives exchange messages from every other node in the neighborhood. So in our implementation we can guarantee that until at least 3 nodes are present in the network, the synchronization process will not fail.

5-3-2 NODE RE(JOIN)

(Re) Joining a new node to the network while performing the synchronization can also cause problems. A simple approach that was previously proposed for the FTSP algorithm (see 5-2-2) is also used in our implementation of RBS.

In our implementation of RBS algorithm, we defined a beacon number to be placed in reference broadcast messages. This field is later used in creation of exchange messages and keeping track of neighbors' beacons and exchange messages. If a beacon message is received by a node from a neighbor with smaller beacon number that was previously observed, it can detect that the neighbor has been restarted.

5-3-3 COMMUNICATION FAILURE

In this subsection we focus on message collisions that can happen due to simultaneous broadcasts of exchange messages. We can either try to detect these kinds of collisions and compensate for them by providing a reliable message delivery or employ a method to avoid collisions.

5-3-3-1 COLLISION AVOIDANCE

Collision management and avoidance are fundamental issues in wireless network protocols and many of the clock synchronization algorithms such as RBS suffer from that.

One of the biggest challenges of RBS is the number of message collisions that can happen due to simultaneous broadcasts. When nodes receive the reference broadcast message, they create a broadcast message to exchange the arrival time of the beacon with other neighbors. Since the event of broadcasting the exchange message is triggered by arrival of the beacon, the probability that the node start broadcasting at almost the same time is really high and this will lead to a high number of collisions.

Table 1 illustrates the number of message losses due to exchange message collisions. Each row shows a different try in which number of exchange messages received for 300 beacons sent and the percentage of lost messages are presented.

Table 1 Message loss due to collision

Test (#)	Number of exchanges Received out of 300 Messages Sent	Message loss (%)
1	154	51.33
2	152	50.66
3	209	69.66
4	128	42.66
5	179	59.66
6	197	65.66
7	199	66.33
8	186	62

In this subsection we describe two simple methods for collision avoidance and present the results of testing them in our sensor network.

5-3-3-1-1 SIMPLE RANDOM BACKOFF

A typical way to avoid collisions due to simultaneous multiple broadcasts is to employ a backoff scheme. With this method, the propagation of data over the medium is delayed by a period of time. This backoff period is typically selected uniformly randomly from a continuous space of numbers. Randomized backoff schemes are very simple; each receiver node should wait for a time which is randomly selected from the backoff space before broadcasts its response. Clearly, the backoff period represents different trade-offs between fault-tolerance, time and energy efficiency. If the backoff period is tuned to reduce the collision rate, the delays in delivering messages will be longer and nodes should keep their radio transceiver on for a longer time. Therefore, the selection of an appropriate backoff space is crucial to the overall performance of the network [21].

In order to tune the backoff period, we added a uniformly selected random delay before broadcasting exchange messages. The random value is calculated using the random_rand() function provided in the random library of the Contiki operating system.

Table 2 illustrates number of message losses due to collision using different backoff periods. In each experiment 300 beacons were sent by a reference node and two other nodes receiving it delayed broadcasting the exchange message for a random time in the backoff space. It can be seen that with this method we can reduce the number of losses from around 40-70% to 19-38%, but collisions are not completely avoided.

Table 2 Random backoff space and number of collisions

Test (#)	Backoff space (ms)	Number of exchange messages Received out of 300 Messages Sent	Message loss (%)
1	[0,0.5]	195	35
2	[0,0.5]	200	33.33
3	[0,1]	243	19
4	[0,1]	226	24.66
5	[0,1]	213	29
6	[0,1]	241	19.66
7	[0,1.5]	193	35.66
8	[0,1.5]	196	34.66
9	[0,1.5]	205	31.66
10	[0,5]	228	24
11	[0,10]	221	26.33
12	[0,10]	223	25.66
13	[0,15]	197	34.33
14	[0,150]	226	24.66
15	[0,1500]	187	37.66

5-3-3-1-2 TDMA-based Scheduling

Another solution that can be adopted to avoid collisions is to emulate behavior of a Time Division Multiple Access (TDMA) MAC protocol on a higher layer. Emulating TDMA scheduling can eliminate collisions and bound the delay [22].

TDMA scheduling allows several nodes to share the wireless media channel by dividing transmission into different time slots. The nodes transmit one after the other using their dedicated time slot. For example assuming that nodes are scheduled according to their unique IDs, a node with ID 1 can broadcast its message in the first time slot in each round, but node with ID 2, should wait for first slot to finish and then broadcasts its message in its own slot.

TDMA algorithms consider either one-hop or multi-hop scheduling. In single hop networks only one node is allowed to transmit in a slot. In multi hop networks, in contrast, more than one node can transmit in a time slot provided that the radio coverage of the receivers has no conflict. Finding a scheduling algorithm that minimizes the number of required slots is a NP-complete problem [22] and is outside the scope of this thesis. In this section we focus on scheduling for one-hop network and present results of applying TDMA scheduling to a network including 4 sensor nodes.

For implementing a TDMA-based scheduling, we divided the time period for each round to *n* slots, where *n* is the number of nodes in a one-hop network. In each round one node broadcasts a beacon and others broadcast an exchange message in response. The first slot in each round is dedicated to the beacon sender. Other slots are dedicated to the rest of nodes according to their node IDs.

Figure 19 depicts the slot allocations for a 4-node network where nodes take turn to send beacons in a round-robin fashion. As an example, consider a 4-node network with IDs from 1 to 4, if node ID 3 is sending the beacon the first slot should be assigned to it. The second slot will be assigned to node ID 1, third slot to node ID 2 and the last slot to node ID 4.

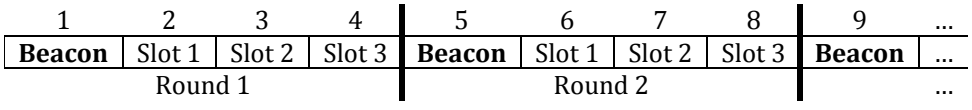

Figure 19 TDMA scheduling slot assignment

Another scheduling strategy that can be employed is to statically assign a slot to each node in the network (i.e. node ID 1 will send in the first slot, node ID 2 in the second slot and so on). In this strategy, each node should piggy-back its

beacon with the exchange messages in response to previously received beacons. So each node should store reception time of its neighbors' beacons and append them to its own beacon message. In this way the number of messages being sent and received will be reduced but the length of the broadcast message will grow with the number of nodes in the network.

There is a trade-off in employing either of these scheduling strategies. In the first strategy messages are smaller so they consume less memory in a resource constrained sensor node, but on the other hand this method produces a large number of messages that may increase the probability of colliding with application generated messages as well as consuming more energy for each time the transmitter is turned on. In the second scheme, by increase of the number of nodes in the network message sizes are raised, imposing the need for larger memory buffers. But by reducing the number of messages the power required for turning on the radio transmitter is conserved. Another problem is that by losing a message due to media noise or transient faults the clock synchronization process is prolonged, leading to lower synchronization precision.

If nodes know the ID of their neighbors prior to clock synchronization, the slot selection becomes trivial and collisions are avoided from the very beginning. But in reality where sensor nodes are deployed randomly and network topologies are dynamic, finding a strategy for starting-up the slot assignment is inevitable.

In the literature several TDMA start-up solutions are proposed. One possible solution is as follows: Each node broadcasts a beacon message, if it didn't receive any exchange message in a round, it detects that either its beacon message has collide or response exchange messages have collide, so it selects another slot for the next round. Slot selection can be achieved either by delaying transmission for a randomly selected period of time or according to any other scheme to reduce the risk of collisions. Upon receipt of each beacon or exchange message, the node checks the sender ID, if the ID is not already in its neighbors list, it is entered there. The node should sort this list to be able to choose its own slot for each round according to that. Unfortunately, with the existence of communication errors, packet used for slot assignment can be dropped, causing the start-up process to converge slowly. In our implementation we used the first TDMA scheduling solution in order to achieve faster synchronization.

As mentioned in previous sections, a sensor network may use clock synchronization for TDMA medium access scheduling, but here TDMA scheduling is employed for achieving better clock synchronization. Although collision among the nodes can be avoided using TDMA scheduling, but message collisions can still occur due to unsynchronized clocks. If each node calculates its slot using its local clock, slots of unsynchronized nodes can

overlap and lead to collisions. Figure 20 illustrates an example of slot overlapping of unsynchronized nodes.

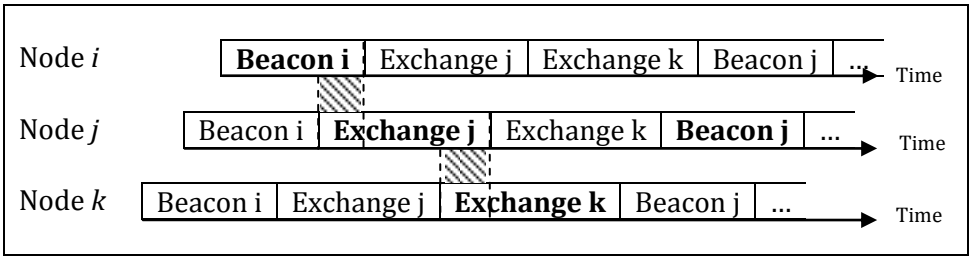

Figure 20 TDMA scheduling without synchronized clocks. Beacon message sent by node *i* may collide with exchange message sent by node *j*, and exchange message sent by node *k* may also collide with exchange sent by *j* in the time periods marked in the figure

Thus, clock synchronization is required to achieve TDMA scheduling and TDMA scheduling can be used for better time synchronization. Therefore, the startup mechanism and the scheduling algorithm must be designed carefully, since messages must be sent to achieve synchronization while nodes have to be synchronized to guarantee collision-free broadcasts.

Although recent analysis of radio transmission characteristics of sensor networks shows that TDMA may not considerably improve the bandwidth in comparison to randomized collision avoidance protocols, but fairness and energy saving considerations are still important motivations for using TDMA [23].

We implemented the described TDMA scheduling scheme to avoid collisions in the RBS protocol. In our experiments each node broadcasted 300 beacons and 900 exchanges in response to received beacons. Experiment results showed that when using local time for slot selection, around 0-0.0044% of exchange messages were lost due to collision. In case of using global time for slot selection we didn't face any collisions after more than 3600 exchange messages being sent and received.

5-4 COMPARING RBS AND FTSP

In section 4-3 we compared the RBS and FTSP clock synchronization approaches according to their precision, message complexity and energy consumption. In this chapter we focused on fault-tolerance issues in these algorithms and proposed solutions. Now we can compare the basic algorithms with their improved versions considering their fault tolerance, number of collisions and complexity in general. Table 3 summarizes the comparisons.

Table 3 Comparison of different versions of RBS and FTSP

Protocol	Node Failure Tolerance	Message Collision	Overall Complexity
Basic RBS (with one beacon sender)	Low	High	Medium
RBS (with all nodes sending beacons)	High	High	High
RBS with random back-off	High	Low	High
RBS with TDMA using local time	High	Low	High
RBS with TDMA using global time	High	Very Low	High
Basic FTSP	Low	None	Low
FTSP with leader election	High	Medium	Medium

It can be seen that both basic RBS and FTSP synchronization approaches are not fault tolerance. However, RBS can be implemented in a completely distributed manner which makes this approach more fault-tolerant and reliable. Unfortunately RBS suffers from large number of message collisions which necessitates implementation of collision avoidance strategies that will increase the complexity. By emulating TDMA scheduling for transmission of exchange messages in the RBS algorithm we could avoid collisions while introducing reasonable complexity.

FTSP has the advantage of being simple and having low complexity. Although this approach does not suffer from message collisions, but as mentioned before, it is not fault tolerant. By implementing a dynamic distributed leader election the single point of failure is removed from the network. However, the leader election procedure increases the complexity of the FTSP and the possibility of collision of synchronization and leader election messages.

CHAPTER 6

SECURITY

Sensor networks are typically deployed in unattended environments that are usually not trusted. In addition, since nodes communicate using a radio channel, all communications are subject to eavesdropping. Therefore, sensor network security can easily be breached by malicious adversaries.

Most of existing clock synchronization protocols developed for sensor networks such as RBS and FTSP assume benign environments and only focus on maximizing the synchronization precision, energy efficiency, scalability and robustness to dynamic topology changing. However, in hostile environments, an adversary may attacks the time synchronization protocol due to its importance by either passive attacks such as eavesdropping, or active attacks such as message injection or denial of service. Unfortunately none of these protocols were designed considering security as a goal. Given the unattended nature of many sensor network deployments and the importance of the applications and services which require time-synchronization, the security of time synchronization should be considered at design time [24].

In this section some malicious attacks and threats to time synchronization protocols as listed in the literature are described briefly.

6-1 THREATS TO TIME SYNCHRONIZATION IN WIRELESS SENSOR NETWORKS

Almost all the attacks on time synchronization protocols have the main goal of convincing some nodes that their neighbors' clocks are at a different time than they actually are [24]. Since all time synchronization protocols rely on time-sensitive message exchanges, adversaries can easily tamper with synchronization protocols by attacking these messages.

Adversaries may launch a *message manipulation attack* [24] in which, the attacker may drop, modify, or even forge the time synchronization messages to mislead the time synchronization process. Attacker can jam the radio channel to launch *Denial of Service* (DoS) attacks. Adversaries can also launch more sophisticated attacks such as *pulse-delay attacks* [24]. In the pulse-delay attack, the adversary snoops messages, jams the receipt of time synchronization messages, and later replays buffered copies of these messages at the his/her choice of time. The adversary may launch Sybil attacks by adding a node that presents multiple identities. Adversaries may compromise some nodes or introduce new nodes to the network and exploit these nodes in arbitrary ways to attack the time synchronization protocol.

Many of these attacks can be addressed by employing appropriate cryptographic techniques. For example, by authenticating every time-sensitive message it is possible to prevent impersonating other nodes or forging the synchronization messages. Or by adding a sequence numbers to messages replay attacks can be avoided. Unfortunately, some attacks such as pulse-delay attacks cannot be addressed by using cryptographic counter measures [8,25].

Manzo etc. [24] and Song etc. [25] outlined some of the possible attacks on several clock synchronization protocols such as RBS and FTSP and proposed some countermeasures for these attacks.

6-1-1 ATTACKS ON RBS

An attacker may launch different kinds of attacks on reference broadcast synchronization (RBS) [3] to break the protocol.

One possible attack is that an attacker node can impersonate one of the nodes in the network and send an exchange message with wrong time information to disrupt the synchronization process. In addition, a compromised node can also send a falsified exchange message to its neighbors. This can lead to incorrect calculations of phase offset and skew by honest nodes. Another possible attack is a replay attack in which the attacker's node can replay a legitimate node's old exchange packet to mislead its neighbors to be synchronized to a wrong time.

Moreover, the attacker may selectively drop some packets by jamming the communication channel and make the synchronization convergence much longer. Adversary can also launch a message forging attack by creating many fake reference beacon messages and flooding the network with them. This attack not only misleads the synchronization process, but also forces the nodes to consume more power to process the forged messages [25].

It has been shown in [24] that if there are large number of nodes in the sensor network, a small fraction of the compromised nodes can cause the estimated global to get far from the true global time. It has been proposed to use least median of squares (LMS) instead of least squares (LS) to fit a more robust model to the stored offset values in the regression table, because LMS is resilient to a high fraction of outliers.

In our implementation of RBS protocol, we try to reject outliers by comparing the offset value calculated from the newly received exchange message with the average of values already stored in the regression table. If the difference is greater than some predefined threshold, that value is rejected and will not affect phase offset and skew calculations. In this way if an impersonated or compromised node sends a falsified exchange message with wrong timestamp, and the value is too far in the past or in the future, it will be discarded.

6-1-2 Attacks on FTSP

In FTSP the root is chosen dynamically. Any node may claim to be the root if it has not heard time updates for a predefined period. One possible attack described in [24] is that a compromised node can claim to be the root node with a small node ID (e.g. ID 0) and broadcast synchronization messages with higher sequence number than the actual root node. Other nodes receiving the synchronization messages from the compromised node will ignore real root's broadcasts. Once the compromised node becomes the root it can send false time to its neighbors so every node that accepts the false updates will calculate an erroneous phase offset and skew.

6-2 Secure Clock Synchronization

Recently many protocols that take security and fault tolerance into account are proposed in the literature. A secure time synchronization protocol should be able to mask attacks launched by adversaries who try to mislead the synchronization process. Many of the existing secure synchronization algorithms employ cryptographic techniques such as authentication to mask malicious attacks [24,25,26]. In the following subsection, we focus on a secure and self-stabilizing algorithm for clock synchronization in sensor networks which is proposed in [8].

6-2-1 SECURE AND SELF-STABILIZING CLOCK SYNCHRONIZATION

The algorithm proposed in [8] is the first secure and self-stabilizing clock synchronization algorithm in sensor networks. A self-stabilizing algorithm guarantees that from any arbitrary starting configuration, the system will accomplish its tasks even in the presence of transient faults [30]. Therefore, this algorithm can ensure automatic recovery after arbitrary failures, in addition to tolerating message omission failures due to collisions or random media noise. The proposed protocol considers fine-grained clock synchronization and focuses on the fault-tolerance aspects of secure clock synchronization protocols.

This algorithm assumes that neighboring nodes can directly communicate with each other by using secure broadcast primitives. Nodes should use predefined pair-wise secret keys to perform symmetric key cryptography. Encryption and adding a message authentication code guarantees confidentiality and message integrity. Also by adding a counter to the message before applying the cryptographic operations and allowing the receivers to reject old messages, messages' freshness can be ensured. Unfortunately, some of the possible attacks on clock synchronization, such as pulse-delay attacks (see section 6-1), have no cryptographic counter measures. But this algorithm is resilient to such adversary attacks even in presence of captured nodes that their secret keys are revealed by the adversary and are impersonated.

In this approach n neighboring clocks are sampled in the presence of t faulty or compromised nodes. The clock sampling algorithm can make it possible to employ different kind of masking techniques such as byzantine agreement or considering statistical outliers to overcome pulse-delay attacks in the presence of captured nodes.

In this clock synchronization protocol, sensor nodes are in communication range of each other and periodically broadcast beacons, respond to received beacons and after aggregating the beacons with their responses, deliver them as a record to the upper layer. In the upper layer, responses to delayed beacons are removed to mask the effect of pulse-delay attacks. This algorithm allows the use of clock synchronization techniques such as reference broadcast (RBS) for skew and offset estimations in the upper layer. So the algorithm itself only focuses on two tasks: *beacon scheduling* and *beacon and response aggregation*.

Beacon scheduling includes broadcasting a beacon and waiting for its response to guarantee round-trip message exchange. Nodes are scheduled for beacon transmission by using a randomized scheduling strategy that with high probability avoids collisions. In this simple scheduling strategy, the nodes

select a random time broadcast from a pre-defined period. Redundant broadcasting timeslots are also used to overcome the clocks' asynchrony.

Beacon and response aggregation task is done when the round-trip exchange is complete by delivering the beacon and its responses to the upper layer. In this algorithm each node maintains a sequence of its most newly sent beacons and arrival time of a number of most recently received beacons. When a correct node is scheduled to act as the synchronizer, it broadcasts a beacon piggy-backed with response messages of a number of most recently received beacons from all other nodes.

Detailed description of the secure and self-stabilizing algorithm and proof of correctness can be found in [8].

CHAPTER 7

APPLICATIONS BASED ON TIME SYNCHRONIZATION

Wireless sensor networks are deployable in different applications in various domains such as military, environmental, medical, industrial, civilian, and home networks. In military domain, sensor networks can be used to prevent enemy attacks by detecting their aircrafts and army or to monitor equipments and friendly forces. In the civilian domain, surveillance for security in harbors, airports, banks, bridges, etc. monitoring and detecting chemical fluid leakage or presence of hazardous materials can be achieved by deploying sensor networks.

In this chapter we consider a surveillance application scenario with the objection of identifying a breach within a protected region, and demonstrate the importance of employing clock synchronization in such applications.

7-1 MOTION DETECTION AND TRACKING

Motion detection applications typically require detecting that a movement exists and should be able to track the moving target. In general, detection requires that the system be able to discriminate between a target's absence and presence [27]. Successful movement detection necessitates that a sensor node correctly estimates a movement while avoiding false detections in which no target is present. The key performance metrics for detection as described in [27] are: the probability of correct detection, the probability of false alarm, and the allowable latency between a targets presence and its eventual detection. Tracking a movement requires maintaining the targets position as it moves over time in a region covered by sensor networks. Successful tracking

requires that the system estimate a targets initial point of entry and current position with sufficient accuracy and with tolerable latency.

By knowing this fact that it is not possible to take into account all of the environmental factors that might affect the system, the selection of sensors becomes an important task in design of the sensor networks. Choosing the right set of sensors for the job at hand can dramatically improve systems performance, lower its cost, and improve its lifetime [27]. However, the output generated by a sensor and the computation resources required for processing it should be taken into account. For example, even a small camera have tens of thousands of pixels that provide an enormous amount of information and often requires significant computational resources which are not available on resource constrained sensor nodes.

For our movement detection scenario, we decided to choose a cheap and small motion detection sensor which only produces a one-bit output.

7-1-1 PIR SENSOR

PIR stands for Passive Infra-Red sensor which is a pyroelectric sensor that detects human body movements up to 6 meters away from itself. Figure 21 demonstrates a Parallex PIR sensor [28] which is 24.3×32.2mm large.

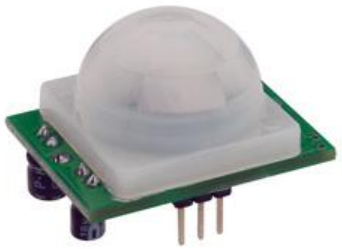

Figure 21 Parallex PIR Sensor

The PIR uses a Fresnel lens and infrared-sensitive IC to detect changing patterns of passive infrared emitted by objects in its vicinity. The motion can be detected by checking for a high signal on a single I/O pin.

Inexpensive and easy to use, it's ideal for alarm systems, motion-activated lighting, holiday props, and robotics applications.

The PIR Sensor requires a 'warm-up' time in order to function properly. This is due to the settling time involved in 'learning' its environment which may takes 10-60 seconds. Since in the first seconds of operation, the output is

continuously turning on and off it is best to make as little motion as possible in the sensors field of view.

As mentioned earlier, the PIR Sensor has a range of approximately 6 meter which can vary depending on the environmental conditions. The sensor is designed to adjust to slowly changing conditions that would normally happen, but when sudden changes occur in the environment such as when there is motion, the sensor reacts by making its output high. If the motion is continuous, the output remains high. After motion stops, the output remains high for from 2-4 seconds.

The useful detection angle of the PIR sensor is about 45 degrees on either side, for a total of 90 degrees. It has been suggested that to control the angle of sensitivity, the sensor could be mounted in a PVC cap with an appropriate length to control its angle.

7-2 SCENARIO

To clarify the impact of clock synchronization in sensor network applications such as motion detection, we conducted a series of experiments. We connected the PIR sensor on top of our MSB-430 sensor nodes to be able to communicate the detected movements to a base station. In our scenario we employed 4 nodes for motion detection and connected one of them to a PC (base station) to act as a Sink node. The nodes where placed 3-4 meters away from each other and their detection angle was reduced by using a cap.

Upon detection of a movement each node sends an alarm message containing its node ID and the time in which the movement was detected towards the sink node. The collected alarm messages are used by an application installed on the base station to track the movement. Figure 22 shows the GUI of the application we implemented for reading the output from the Sink node.

In real world applications the sensor nodes are turned on with arbitrary orders that will introduce initial phase offsets. Even if we manage to start all the nodes exactly at the same time, the clocks of the nodes will drift away because of differences in frequency of clock oscillations.

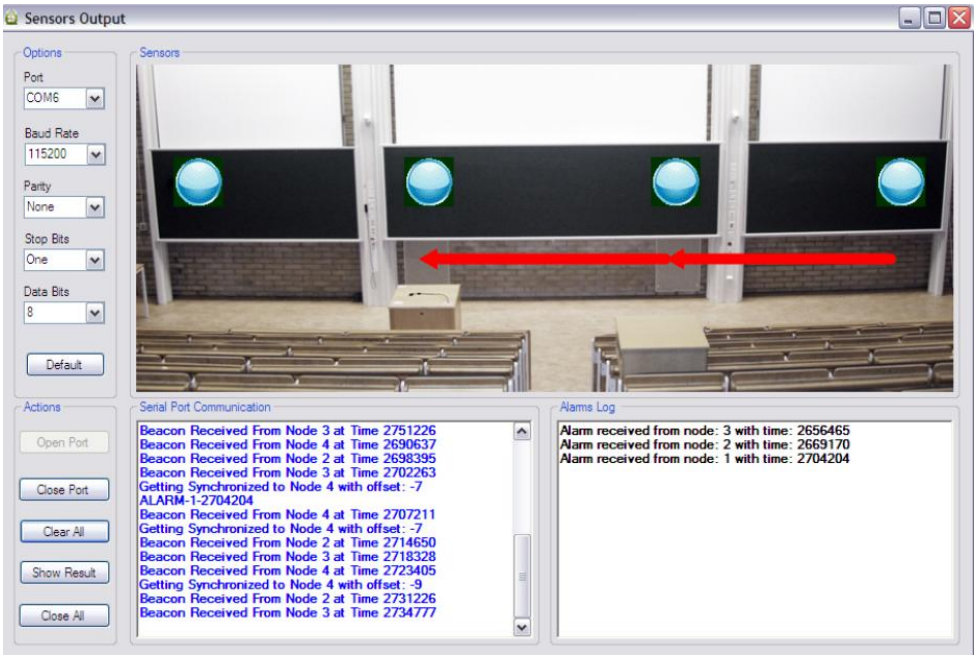

Figure 22 Sensor network's base station GUI

Even from the first experiment it becomes clear that after ordering the timestamps of the received alarm messages, it is possible that a wrong direction is detected. In our scenario we started the nodes one by one to introduce initial offsets. An intruder passes in front of the sensors and its movement is detected correctly by nodes and is sent to the Sink node. When the base station receives the alarms, it tries to track the direction of the movement by sorting the time values reported by the sensors. Since nodes' clocks are not synchronized the detected direction can significantly differ from the actual one. By starting the synchronization algorithm the possibility of detecting wrong direction is dramatically reduced.

For this scenario we set the synchronization period to be 1 second. Previous experiments in chapter 3 showed, the maximum absolute synchronization error between two nodes with this synchronization period was 9 clock ticks in the RBS algorithm. Assuming that the sensor nodes are placed at most 12 meters away from each other and the movement detection sensors are ideal, the estimated global time by 2 neighboring nodes can drift away at most 2.2ms. It means that if a moving target passes the sensors with speeds higher than around 5400m/s, there is a possibility of detecting incorrect movement direction. Considering speed of a typical rifle bullet which varies between 180 and 1500 meters per second[3], it is clear that the synchronization algorithm error cannot adversely affect the detection and tracking application.

[3] http://hypertextbook.com/facts/1999/MariaPereyra.shtml

To test how fault tolerant is our synchronization algorithm, the adversary steals one of the sensor nodes. Removing a node from network will not affect the synchronization procedure because the other nodes take turn to broadcast reference (beacon) messages.

When a new node is added to existing network, the rest of the nodes detect its presence by checking the beacon number of newly arrived beacons. The new node adopts itself to the network by adjusting its clock counter value. The synchronization process continues to work and the newly joint node will synchronize after gathering enough exchange messages. In our scenario after at most 2 seconds the new node receives exchange messages from its neighbors and can synchronize itself. To achieve higher precisions, the new node should collect more synchronization points in its regression table, for example 8 data points. So the required synchronization precision will be achieved after only $8 \times T$ seconds (where T is the synchronization period), which is much smaller than the calibration time (10-60s) required for the sensors to be able to start detection.

These experiments clarify the importance of employing a fine-grained clock synchronization algorithm, particularly in applications that depend on common notion of time. The importance of clock synchronization is even more crucial in large networks where TDMA radio scheduling is used or where alarm messages are aggregated before arriving at the base station. In these situations, messages are not received with the same order that the events generating them has happened. Therefore the only way that events can be ordered is by using the timestamp values embedded in them.

Chapter 8

CONCLUSION AND FUTURE WORK

8-1 CONCLUSION

In this thesis we show the importance of clock synchronization for sensor network applications and the possibility of implementing fine grained synchronization on sensor nodes. We select two well-known approaches for clock synchronization named FTSP and RBS.

FTSP is a leader-based synchronization scheme that allows nodes in a network to get synchronized to a transmitter node. This approach takes advantage of time-stamping at lowest layer of the communication stack and provides precise synchronization. On the other hand, RBS is a reference broadcast synchronization scheme that synchronizes a set of receivers with one another. This algorithm provides a fine-grained synchronization and can be implemented in a completely distributed manner.

In this thesis we implement these approaches on our MSB-430 platform using Contiki operating system and evaluate our experiment results. We also show the possibility of implementing robust and fault tolerant synchronization algorithms that tolerate node failures, node (re)joins and messages losses. In our implementations nodes in the network adopt themselves to node failures and node (re)joins. We proposed the use of a TDMA-based scheduling to avoid collisions of synchronization messages.

We also described some of the possible security attacks against time synchronization and reviewed some of the proposed countermeasures in the literature including a secure and self-stabilizing synchronization approach. This approach can be employed together with a synchronization algorithm

such as RBS to provide a fine-grained synchronization that tolerates pulse-delay attacks in presence of captured nodes.

Finally our implemented fault-tolerant clock synchronization is tested in a real security surveillance application to detect and track movements of an intruder. Our experiments show the importance of employing fine-grained synchronization in such applications and show how the algorithm can tolerate node failures and network extensions.

8-2 Future Work

In future the presented fault-tolerant clock synchronization approaches could be extended to larger multi-hop networks. In [4] a multi-hop version of FTSP is introduced. In this approach a synchronization-root is dynamically elected and the whole network is being synchronized to it. The synchronization message produced by the root is flooded through the network. Therefore, nodes in communication range of the root directly synchronize to the root, but nodes further away indirectly synchronize themselves through nodes that are closer to the root. This approach gives good synchronization precision but introduces many reliability and security issues.

Since all synchronized nodes should periodically transmit synchronization messages to flood the network, the probability of message collision is very high. Also if the synchronization root fails, the leader election procedure messages will flood the network. By capturing or impersonating nodes closer to the root, an adversary can corrupt the synchronization of clocks of further nodes. It is suggested in [24] to add redundancy to this scheme by recording a subset of synchronization messages from their neighbors.

In [3] an extension to basic RBS is also proposed to enable synchronization in a multi-hop network. In this approach a “time route” is computed to dynamically convert timestamps from one clusters’ time-base to another. The conversion is done by intermediary nodes that are placed in overlapping regions of single-hop clusters. Although each conversion will add synchronization errors, the authors have shown that synchronization across many hops will not significantly degrade the precision [13]. As described in [24], if an adversary can compromise a node in an overlapping region to produce erroneous clock conversions, all the overlapping regions will be affected. Once a wrong conversion is sent, the error will be propagated throughout the network.

Therefore, extending a robust and fault-tolerant or secure and self-stabilizing synchronization approach to larger networks is a challenging issue that is an active research area.

We are also planning to perform further experiments on secure and self-stabilizing synchronization algorithm to verify its performance in presence of captured nodes.

An implementation issue that can be of interest for future works, is employing Contiki's timer that counts the processor cycles executed by the CPU of the nodes. We tried to use this timer which could give us a much faster clock (2.4576 MHz) with finer resolution. Surprisingly the synchronization precision achieved using this timer was much worse than the simple timer which counts the quartz crystal oscillations. For example the average and maximum absolute error values for basic FTSP implementation was 44616.23 CPU cycles (18154μs) and 126402.9 CPU cycles (51433μs) respectively (see section 4-2). More research on how this clock works and how we can employ it in a way that improves synchronization precision can be done in future works.

Another implementation problem we faced that can be looked into in future was using multiple rtimers in a single process. For implementation of RBS algorithm with TDMA we need to send exchange messages in the nodes' dedicated time slot. The best way to make sure that the message will be sent in appropriate time is to set an rtimer to schedule the send task at the beginning of time slot. Meanwhile we need to count the number of times the system clock wraps around. The only way to perform this task accurately is to set another rtimer to schedule a task at the exact time the clock value changes from 65535 to zero to increment the counter. Unfortunately, there is a bug in the rtimer module of the Contiki operating system that causes problems when using multiple concurrent rtimers[4].

Finally, optimizing the period of sending synchronization messages for energy efficiency is also an interesting challenge to look into in the future. We should study that with which frequency the synchronization messages should be sent to make sure the required precision is preserved while trying to conserve energy and avoid colliding with application messages. One possible solution is to initiate the synchronization procedure with small synchronization period to guarantee fast deployment, and after achieving the necessary precision the period can be increased. This solution needs more study to ensure that application requirements will be met even in case of failures or utilizing new nodes in the network.

[4] http://www.sics.se/contiki/changelog.html

BIBLIOGRAPHY

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Computer Networks*, vol. 38, pp. 393-422, 2002.

[2] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock synchronization for wireless sensor networks: a survey," *Ad Hoc Networks*, vol. 3, pp. 281-323, 2005.

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

[4] M. Marti, B. Kusy, G. Simon, and Akos Ledeczi, "The flooding time synchronization protocol." In *SenSys*, J. A. Stankovic, A. Arora, and R. Govindan, Eds., pp. 39-49, 2004.

[5] ScatterWeb Homepage, Freie Universität Berlin, Berlin 2008, http://scatterweb.mi.fu-berlin.de

[6] Contiki Homepage, Swedish Insitute of Computer Science, Stockholm 2008, http://www.sics.se/contiki

[7] A. Dunkels, F. Sterlind, and Z. He, "An adaptive communication architecture for wireless sensor networks," in *Proceedings of the Fifth ACM Conference on Networked Embedded Sensor Systems (SenSys2007)*, 2007.

[8] J. H. Hoepman, A. Larsson, E. M. Schiller, and P. Tsigas, "Secure and self-stabilizing clock synchronization in sensor networks," In the *Proceedings of the 9th International Symposium on Self Stabilization, Safety, And Security of Distributed Systems (SSS 2007), Lecture Notes in Computer Science Vol.:* 4838, pages 340-356, Springer-Verlag 2007.

[9] T. Herman and C. Zhang. “Best paper: Stabilizing clock synchronization for wireless sensor networks.” In A. K. Datta and M. Gradinariu, editors, SSS, volume 4280 of LNCS, pages 335–349. Springer, 2006.

[10] H. Kopetz and W. Ochsenreiter, “Clock synchronization in distributed real-time systems,” *IEEE Transactions on Computers*, no. 36, pp. 933-939, 1987.

[11] J. Mannermaa, K. Kalliomaki, T. Mansten, and S. Turunen, “Timing performance of various gps receivers,” *Frequency and Time Forum, 1999 and the IEEE International Frequency Control Symposium, 1999., Proceedings of the 1999 Joint Meeting of the European*, vol. 1, pp. 287-290 vol.1, 1999.

[12] D. Mills, “Internet time synchronization: the network time protocol,” *Communications, IEEE Transactions on*, vol. 39, no. 10, pp. 1482-1493, Oct 1991.

[13] Jeremy Elson, “Time Synchronization in Wireless Sensor Networks,” *Ph.D. dissertation*, University of California, Los Angeles. May 2003.

[14] F. Cristian, “Probabilistic clock synchronization,” *Distributed Computing*, vol. 3, no. 3, pp. 146-158, 1989.

[15] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” *Commun. ACM*, vol. 21, no. 7, pp. 558-565, July 1978.

[16] S. Ganeriwal, R. Kumar, and M. B. Srivastava, “Timing-sync protocol for sensor networks.” ACM Press, pp. 138-149, 2003.

[17] Chipcon, CC1020 datasheet, Chipcon AS, Oslo 2005.

[18] A. Dunkels, B. Grönvall and T. Voigt, “Contiki – a Lightweight and Flexible Operating System for Tiny Networked Sensors,” *In Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks table of contents*, p.p. 455-462, 2004.

[19] A. Dunkels, “Using protothreads for sensor node programming,” *In Proceedings of the REALWSN 2005 Workshop on RealWorld Wireless Sensor Networks*, 2005.

[20] A. Dunkels, “The uip embedded tcp/ip stack. The uip 1.0 reference manual”. *Technical report*, Swedish Institute of Computer Science, 2006.

[21] I. Chatzigiannakis, A. Kinalis, and S. Nikoletseas, “Wireless sensor networks protocols for efficient collision avoidance in multi-path data propagation,” in *PE-WASUN ’04: Proceedings of the 1st ACM international workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, pp. 8-16, 2004.

[22] S.C. Ergen, P. Varaiya, "TDMA scheduling algorithms for sensor networks," *Technical Report*, Department of Electrical Engineering and Computer Sciences University of California, Berkeley, July, 2005.

[23] T. Herman and S. Tixeuil. "A distributed TDMA slot assignment algorithm for wireless sensor networks." *In Proceedings of the First Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors'2004)*, number 3121 in Lecture Notes in Computer Science, pages 45-58, Turku, Finland, July 2004.

[24] M. Manzo, T. Roosta, and S. Sastry. "Time synchronization attacks in sensor networks." *In Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks (SASN'05),* pages 107-116, NYC, NY, USA, 2005.

[25] H. Song, S. Zhu, and G. Cao. "Attack-resilient time synchronization for wireless sensor networks." *Ad Hoc Networks,* 5(1):112-125, 2007.

[26] S. Ganeriwal, S. ·Capkun, C.-C. Han, and M. B. Srivastava, "Secure time synchronization service for sensor networks," in *WiSe '05: Proceedings of the 4th ACM workshop on Wireless security*, 2005, pp. 97-106.

[27] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita, "A line in the sand: A wireless sensor network for target detection, classi¯cation, and tracking,"*Computer Networks (Elsevier*, vol. 46, pp. 605-634, 2004.

[28] Parallax Homepage, 2009, http://www.parallax.com

[29] K. Römer, P. Blum, and L. Meier. "Time synchronization and calibration in wireless sensor networks." In I. Stojmenovic, editor, *Handbook of Sensor Networks: Algorithms and Architectures*, pages 199–237. John Wiley and Sons, Sep. 2005.

[30] M. Papatriantafilou, P. Tsigas "Self-Stabilizing Wait-Free Clock Synchronization," *Parallel Processing Letters*, 7(3), pages 321-328, 1997, World Scientific Press.

[31] S. Fikret and Y. Bülent, "Time Synchronization in Sensor Networks: A Survey," *IEEE network*, vol. 18, no. 4, pp. 45-50, 2004.

[32] S. Ganeriwal, S. Capkun, Ch. Han, and M. B. Srivastava. "Secure time synchronization service for sensor networks." *In Proceedings of the 4th ACM workshop on Wireless security (WiSe'05),* pages 97-106, NYC, NY, USA, 2005. ACM Press.

[33] K. Sun, P. Ning, and C. Wang. “Secure and resilient clock synchronization in wireless sensor networks.” I*EEE Journal on Selected Areas in Communications*, 24(2):395-408, Feb. 2006.

[34] D. Culler, D. Estrin and M. Srivastava, “Guest Editorsapos; Introduction: Overview of Sensor Networks,” *Computer*, vol. 37, Issue 8, pp. 41–49, Aug. 2004.