

Wireless Embedded Systems: Time, Location, and Applications

Doctoral Thesis**Author(s):**

Sommer, Philipp Alexander

Publication date:

2011

Permanent link:

<https://doi.org/https://doi.org/10.3929/ethz-a-006712401>

Rights / license:

[In Copyright - Non-Commercial Use Permitted](#)

Originally published in:

TIK-Schriftenreihe 128

DISS. ETH NO. 19901
TIK-Schriftenreihe Nr. 128

Wireless Embedded Systems: Time, Location, and Applications

A dissertation submitted to

ETH ZURICH

for the degree of
Doctor of Sciences

presented by

Philipp Alexander Sommer

MSc ETH EEIT, ETH Zurich

born December 29, 1982

citizen of
Sumiswald BE

accepted on the recommendation of

Prof. Dr. Roger Wattenhofer, examiner
Prof. Dr. Ákos Lédeczi, co-examiner
Prof. Dr. John Stankovic, co-examiner

2011

Abstract

Wireless embedded systems combine sensors and actuators with processing, storage and communication capabilities. Networks of small, low-power wireless devices, so called nodes, offer the potential to collect observations of the physical world at unprecedented fidelity and scale.

Time and *location* are of fundamental importance in the context of wireless sensor networks. Accurate time and location information are crucial for many tasks, e.g., sensor data fusion, spatiotemporal coordination, low-power operation, and wireless medium access.

In the first part of this dissertation, we study the problem of *time synchronization*. As clock sources in wireless embedded systems often exhibit severe *drift*, and message exchange is subject to a *delay*, sophisticated algorithms are mandatory to keep clocks in synchronization. Existing time synchronization algorithms are designed to provide *network-wide* synchronization, i.e., between arbitrary nodes in the network. However, as we demonstrate in this dissertation, large synchronization errors may become visible with increasing distance from a reference node, but also close-by nodes in a network may be synchronized poorly. To address these issues, we propose two clock synchronization protocols: the *Gradient Time Synchronization Protocol* to achieve *local* synchronization, and the *PulseSync* protocol for *global* synchronization. We evaluate the performance of both protocols by simulations and experiments in different wireless sensor network testbeds.

The second part of this thesis is dedicated to the question how we can provide accurate location information in the context of wireless embedded systems. We present the *SpiderBat* platform, which provides *node localization* for sensor nodes using ultrasound pulses. By employing multiple transmitters and receivers, we can estimate the distance and angle between neighboring nodes. Using both distance and angle information allows to localize nodes with an accuracy of a few centimeters, even in sparse networks where other approaches fail.

In the third part, we discuss design considerations for the architecture of future wireless embedded systems, as they have evolved from pure research tools to ubiquitous smart things. To this end, we present a resource-oriented approach to facilitate the connection between heterogeneous devices and services, and a novel wearable sensor platform that utilizes mobile phones for personalized sensing applications.

Zusammenfassung

Drahtlose, eingebettete Systeme vereinen Sensoren und Aktoren mit Möglichkeiten für Verarbeitung, Speicherung und Kommunikation. Netzwerke bestehend aus kleinen, stromsparenden, drahtlosen Geräten, sogenannte Sensorknoten, eröffnen neue Möglichkeiten zur Beobachtung der physikalischen Welt in bisher unerreichter Genauigkeit und Ausmass. *Zeit* und *Ort* sind von fundamentaler Bedeutung im Kontext drahtloser Sensornetzwerke. Präzise Zeit und Ortsinformation sind entscheidend für zahlreiche Aufgaben, beispielsweise zur Fusion von Sensordaten, raum-zeitlicher Koordination, energiesparendem Betrieb und drahtlosem Kanalzugriff. Im ersten Teil dieser Dissertation beschäftigen wir uns mit dem Problem der Zeitsynchronisation. Da Uhren in eingebetteten, drahtlosen Systemen oft eine starke Drift aufweisen und der Austausch von Nachrichten Verzögerungen unterliegt, bedarf es technisch ausgereifter Algorithmen, um die Synchronisation der Uhren zu erhalten. Bestehende Algorithmen zur Zeitsynchronisation sind darauf ausgerichtet beliebige Knoten in einem Netzwerk miteinander zu synchronisieren. Diese Arbeit legt dar, dass mit grösser werdendem Abstand von einem Referenzknoten, aber auch zwischen benachbarten Knoten, grosse Ungenauigkeiten in der Synchronisation auftreten können. Um diese Probleme zu beheben schlagen wir zwei Synchronisationsprotokolle vor: das “Gradient Time Synchronization Protocol” für lokale Synchronisation und das “PulseSync”-Protokoll für globale Synchronisation. Wir untersuchen das Verhalten beider Protokolle mittels Simulationen und Experimenten in verschiedenen Testumgebungen für drahtlose Sensornetzwerke.

Der zweite Teil dieser Arbeit behandelt die Fragestellung, wie präzise Ortsbestimmung im Kontext drahtloser, eingebetteter Systemen erreicht werden kann. Wir stellen die Plattform *SpiderBat* vor, welche mit Hilfe von Ultraschall die exakte Position von Sensorknoten bestimmt. Durch Verwendung mehrerer Sender und Empfänger können wir sowohl die Distanz als auch den Winkel zwischen benachbarten Knoten bestimmen. Der Einsatz dieser Informationen erlaubt die Lokalisierung von Knoten mit einer Genauigkeit von wenigen Zentimetern, auch bei geringer Verteilungsdichte wo andere Verfahren misslingen.

Im dritten Teil diskutieren wir anhand welcher Gesichtspunkte die Architektur von zukünftigen drahtlosen, eingebetteten Systemen entworfen werden soll, da sich diese von reinen Forschungsobjekten zu allgegenwärtigen intelligenten Objekten entwickelt haben. Dazu stellen wir einen Ansatz vor, welcher den Zusammenschluss von verschiedenartigen Geräten und Dienstleistungen vereinfacht. Ausserdem präsentieren wir eine neue, tragbare Sensorplattform, welche Mobiltelefone für personalisierte Sensorapplikationen nutzt.

Acknowledgments

Looking back at an exciting time as a Ph.D. student at ETH Zurich, I appreciate that many people supported me during my studies in various ways. First of all, I would like to express my gratitude to my advisor Prof. Dr. Roger Wattenhofer for his guidance and support. I would also like to thank my co-examiners Prof. Dr. Ákos Lédeczi and Prof. Dr. John Stankovic for their willingness to review this thesis and to serve on my committee board.

Furthermore, many thanks go to all current and former colleagues of the Distributed Computing Group at ETH Zurich, namely Nicolas Burri, Raphael Eidenbenz, Yuval Emek, Roland Flury, Olga Goussevskaia, Stephan Holzer, Barbara Keller, Michael Kuhn, Tobias Langner, Christoph Lenzen, Thomas Locher, Roland Mathis, Remo Meier, Yvonne Anne Pignolet, Pascal von Rickenbach, Stefan Schmid, Johannes Schneider, Jochen Seidel, Benjamin Sigg, Jasmin Smula, and Samuel Welten.

Special thanks for many fruitful discussions and technical advice go to all people working with wireless sensor networks at our institute, namely Jan Beutel, Bernhard Buchli, Federico Ferrari, Tonio Gsell, David Hasenfratz, Roman Lim, Matthias Keller, Andreas Meier, Olga Saukh, Christoph Walser, Matthias Woehrle, Mustafa Yucel, and Marco Zimmerling.

A number of internships, semester or master theses have been conducted in the context of this dissertation. I would like to thank: Daniel Burgener, Sebastian Cartier, Fabian Dreier, Thomas Fahrni, Beat Gebistorf, David Gugelmann, Heidi Gumpert, Andreas Huber, Richard Huber, Michael von Kaenel, Michael Meier, Silvan Nellen, Georg Oberholzer, Samuel Pfaffen, Lars Schor, Oliver Schultes, Christian Stocker, Urban Suppiger, Felix Sutton, and Martin Zoller.

Last but not least, I would like to express my gratitude to my family and friends for supporting me during the last four years. I am indebted to my parents Susanne and Ernst for providing me the education I have received. Sadly, my mother could not witness my graduation, but I know that I would not have come so far without her encouragement. I thank Patrick for being the best brother one could wish. Finally, my deepest thanks belong to my girlfriend Ruth for her love and encouragement.

Contents

1	Introduction	4
1.1	Thesis Overview	6
I	Clock Synchronization	8
2	Introduction to Clock Synchronization	9
2.1	History of Time Keeping and Transfer	9
2.2	The Clock Synchronization Problem	12
2.3	Hardware Clocks	13
2.4	Message Delay	16
2.5	Related Work	21
3	Gradient Clock Synchronization	26
3.1	System Model	27
3.2	Hierarchical Clock Synchronization	29
3.3	Gradient Synchronization Algorithm	30
3.4	Simulations	34
3.5	TinyOS Implementation	34
3.6	Experimental Evaluation	37
4	Global Clock Synchronization	47
4.1	Multi-Hop Clock Synchronization	47
4.2	Linear Regression	48
4.3	The Flooding Time Synchronization Protocol	52
4.4	The PulseSync Protocol	54
4.5	Simulations	56
4.6	Experimental Evaluation	56
4.7	Protocol Improvements	65
5	Conclusions	69

II	Node Localization	71
6	Introduction to Localization	72
6.1	A Brief History of Navigation Techniques	73
6.2	Localization in Wireless Embedded Systems	74
7	Localization with Ultrasound	78
7.1	The SpiderBat Platform	79
7.2	Ultrasound Ranging	83
7.3	Angle of Arrival Estimation	87
7.4	Multi-Hop Positioning	91
7.5	Experimental Evaluation	94
7.6	Non Line-of-Sight Propagation	97
8	Conclusions	100
III	Applications	102
9	Architectures for Sensor Networks	103
9.1	Related Work	104
9.2	System Architecture	105
9.3	Implementation	108
9.4	Applications for Smart Buildings	113
10	Personalized Sensing Applications	116
10.1	The Mobile Phone as a Sensing Platform	117
10.2	The Sundroid System	118
11	Conclusions	128

1

Introduction

Many research disciplines depend on the observation of natural phenomena to advance our knowledge and understanding of nature. However, gathering measurement data with high fidelity over a large spatial extent and time period requires significant technical and personal resources. Recent technological advances have led to the miniaturization of scientific instruments such as accelerometers, magnetometers, and temperature sensors. Moreover, computing platforms necessary to acquire and process sensor readings have become down-sized tremendously over the last decades in terms of their costs, form factor and energy consumption.

Wireless embedded systems combine sensors, local storage and processing capabilities with a wireless transceiver to communicate with nearby nodes [93]. By forming a wireless multi-hop network consisting of small battery-powered devices, nodes collaborate to deliver sensor readings to a base station. Wireless sensor nodes can be placed at specific locations of interest independent of any existing power or communication infrastructure, which greatly simplifies their deployment and makes them less intrusive.

Wireless sensor networks (WSNs) have been successfully applied in several domains, such as habitat monitoring [145] and environmental monitoring [150, 149, 7, 9]. Furthermore, anticipated applications of wireless sensor networks also extend into surveillance [48, 148], precision agriculture [77], civil infrastructure [65], building automation [62, 27], health care [158, 88, 107], and people-centric sensing [17].

Without doubt, services for *time* and *location* are inseparable building blocks in distributed systems, and particularly in wireless sensor networks. Without accurate time and location information, sensed data often lose valuable context. Although there are applications where the “when and where” of the sensed data are of no great concern, a majority of applications require to tag the measured data with a timestamp and location. Such a timestamp will only be meaningful if the nodes in the wireless sensor network manage to have an adequate agreement of time. Accurate time is particularly important, first and foremost because of energy efficiency. In order to increase the network lifetime, nodes try to minimize their energy consumption by sleeping whenever possible. Depending on the sampling rate requirements of a specific application, nodes will only wake up sporadically to read a sensor sample, e.g., once every hour. Moreover, nodes must wake up periodically to communicate with their neighbors. Although one can proceed on several ways to establish communication even without accurate clock synchronization, the most energy efficient protocols let two neighboring nodes wake up at exactly the same time. The neighbor nodes can make sure very quickly that no messages have to be exchanged, and go back to sleep. If they need to exchange information, they will go back to sleep as soon as finished. In such a scheme, an accurate time helps to save energy by shortening the necessary wake-up guard times. Furthermore, accurate time can be beneficial to schedule access to the wireless channel by multiple nodes without interference.

Providing a common notion of time in wireless sensor networks can be achieved in several ways. For instance, we can equip all nodes with a receiver for the Global Positioning System (GPS), which provides access to the Coordinated Universal Time (UTC). However, many sensor applications will be deployed indoors, or at other locations without a line-of-sight to the GPS satellites. Furthermore, GPS receivers are expensive in terms of costs and energy consumption.

Another approach is to equip each node with its own local hardware clock to keep track of time. However, no matter how well these hardware clocks will be calibrated at deployment, the clocks will ultimately exhibit a large drift. To allow for an accurate common time, nodes execute a *clock synchronization* algorithm: Nodes communicate by regularly exchanging messages, trying to minimize the synchronization error between the nodes in the network. To this end, a variety of clock synchronization algorithms tailored for wireless sensor networks have been proposed [35, 46, 91, 121]. Achieving accurate synchronization is a challenging problem due to several issues: Apart from the drifting clocks, synchronization is hindered by the fact that the time to send, receive, and process messages cannot be determined exactly. Furthermore, clock drifts change over time, e.g., the most prominent factor here is temperature. This implicates that nodes must periodically be re-synchronized, and that the system has to detect variations in clock speeds

and adapt quickly. Unfortunately, both goals contradict the desire to save energy. Therefore, an efficient protocol cannot simply enforce the best possible synchronization; rather it has to optimize the trade-off between error bounds and communication costs.

Similarly, knowledge about the precise *location* of nodes is a key requirement to gain useful information from sensor measurements. While the position of a node is often known at deployment and remains fixed during the whole operating phase, sensor nodes might also be mobile, e.g., if they are attached to a robot. Furthermore, location information may be beneficial in protocol design. For instance, it may be used in the network layer, to facilitate routing decisions by means of a geographic routing algorithm, or it may be used in the link layer, for controlling interference by means of geographic information. Obtaining an accurate estimation of the current position has been studied extensively for wireless networks. Thereby, most existing approaches exhibit a tradeoff between their accuracy and limitations in terms of costs, energy consumption and form factor. Furthermore, the position of a node may not be determined directly, but only relatively by using the distance or angle to an anchor node with known position.

1.1 Thesis Overview

In the first part of this thesis, we will study two related aspects of the clock synchronization problem: minimizing *local* and *global* clock errors. We commence with an introduction to clock synchronization in wireless sensor networks in Chapter 2. In the following, we focus on providing accurate clock synchronization between nodes that are neighbors in a wireless network. To this end, we propose and evaluate the *Gradient Time Synchronization Protocol (GTSP)* in Chapter 3, which employs a fully distributed clock synchronization algorithm tailored to provide good synchronization between neighbors while allowing larger clock errors between distant nodes. Furthermore, we tackle the problem of providing accurate global synchronization in networks with large diameters. In contrast to the approach presented in Chapter 3, we now assume that nodes should synchronize to the clock of a reference node. To this end, we identify critical weaknesses of current synchronization schemes in Chapter 4 and propose the *PulseSync Protocol* based on these findings. Both protocols are implemented in TinyOS and evaluated on several WSN testbeds. Extensive simulations provide insight on the clock synchronization accuracy on network topologies that are significantly larger than current testbeds.

The second part of this dissertation is dedicated to *location* awareness in wireless embedded systems. We give an overview of different approaches and discuss their applicability in Chapter 6. Following these findings, we propose *SpiderBat*, a novel ultrasound ranging platform, which provides ac-

curate distance and angle measurements on mote-class hardware. We argue that the use of both distances and angles provides an advantage over existing approaches, especially in sparse networks. Furthermore, using multiple ultrasound transmitters and receivers provides additional possibilities, e.g., it is possible to detect multi-path propagation.

In the third part of this thesis, we evaluate the software and hardware architecture for resource-oriented wireless sensor networks and personalized sensing applications. In Chapter 9, we present an approach to access sensor data within a network of heterogeneous devices using resource identifiers known from the Internet. Thereby, nodes communicate using standard Internet protocols that are tailored to resource-constrained devices. Finally, we present an architecture to integrate mobile sensors into a personalized sensing system in Chapter 10. As a proof of concept, we implemented the *Sundroid* system, which provides personalized solar radiation measurements using a wearable sensor connected to the mobile phone of the user.

Part I

Clock Synchronization

2

Introduction to Clock Synchronization

Time plays an important role in the context of wireless embedded systems. Starting with a brief history of general time keeping and time transfer, we introduce the clock synchronization problem in the context of wireless sensor networks. Furthermore, we will give an overview of hardware clocks and on the fundamentals of message exchange. Finally, this chapter concludes with an overview of related work on clock synchronization.

2.1 History of Time Keeping and Transfer

In this section, we give a brief introduction into the history of human time keeping from simple sundials to the modern atomic clock. Furthermore, we will show how time can be transferred from one place to another, and how these methods have been enhanced over time.

For a detailed introduction into the history of timekeeping and cultural aspects of time, we refer to the books on which this section is based: “Revolution in Time: Clocks and the Making of the Modern World” by David S. Landes [75], “The Story of Time” by Kristen Lippincott et al. [86], and “Splitting the Second: The Story of Atomic Time” by Tony Jones [63].

2.1.1 Measuring Time

For thousands of years humans relied on a time scale defined by the sun's apparent motion in the sky. Time has been told by the position of the shadow cast by an object. Obelisks were used as sundials by the ancient Egyptians as early as 3500 BC [2]. The solar day is defined as the time interval between the sun passing through the zenith at noon on two consecutive days. Due to the elliptical orbit of the earth around the sun and the tilt of the earth's axis, the length of a solar day is not constant over the year but varies slightly.

As long as people lived in small rural places with agriculture as their livelihood and traveling was done rather seldom, the mean solar time did serve its purpose just fine. However, the rise of cross-country railway connections and the spread of the telegraph made it necessary to agree on a common time across different states. In 1883, time zones were introduced, first in the U.S. and Canada, followed by an international system of 24 time zones covering the whole world. This so-called *Universal Time* is based on the mean solar time at the Royal Observatory in Greenwich near London.

Up to now, clocks have mainly been used to divide the solar day into smaller units of time, e.g., the second has been defined as $\frac{1}{86,400}$ part of the day. Basically, every modern clock is a combination of two main blocks: an oscillator and a counter. The oscillator produces a periodic signal such as a metronome produces a rhythmic beat. The counter is the element that keeps track of the number of repetitions of the oscillators. Galileo discovered that a swinging pendulum produces a stable frequency which depends on the length of the pendulum itself. Based on this principle, Christiaan Huygens built the first pendulum clock in 1656. A mechanical pendulum clock with two pendulums, a master and a slave connected by an electrical circuit, was invented by William Shortt in 1921. The Shortt clock was used for highly accurate time keeping at research laboratories. Later on, it was replaced by quartz clocks, which provide even better accuracy. Such a clock uses a crystal quartz tuning fork as its oscillating element. Depending on its exact shape and size, a crystal quartz¹ can be made oscillating at its resonance frequency by applying a voltage. A typical resonance frequency is 32,768 kHz, which can be used to generate a stable 1 Hz clock for wrist watches after dividing it by 2^{15} .

Finally, the invention of the Caesium-based atomic clock in 1955 started a new era in time keeping. An atomic clock is based on the electromagnetic signal emitted when a quantum transition between different energy levels takes place in atoms [2]. In 1972, the International Atomic Time (TAI) was introduced, which is based on a weighted average of a large number of atomic clocks all over the world. UTC (Coordinated Universal Time) is closely linked to TAI but contains several "leap seconds" for correction [63].

¹A quartz consists of silicon dioxide (SiO_2), which is also known as silica.

2.1.2 Time Transfer

Establishing a common standard time immediately leads to a new problem. How can we make sure that different clocks show the same time? Clearly, we need a mechanism to keep two clocks in synchronization for as long as possible. This is fairly easy to achieve if the two clocks are placed right next to each other. But how can we disseminate time from a reference clock to another clock located at a different place? Transferring a time or an event from one place to another is not possible without introducing a delay that depends on the propagation speed of the time carrier (e.g., light, sound or radio waves). If we know the distance between the two clocks, and thus also can estimate the corresponding propagation delay, we can synchronize the two clocks accurately². A better approach would be to accurately measure the delay rather than estimating it only. This can be achieved when the incoming signal is sent back immediately or after a constant delay. The sender of the original signal is able to calculate the delay based on the interval between the first transmission and the reception of the second signal. This approach is known as *sender-receiver synchronization*. While this two-way transfer allows to increase the accuracy of the synchronization since the delay can be determined precisely, it works only for a single receiver and does not scale well. Another approach to synchronize clocks is based on a common event that can be detected at multiple locations roughly at the same time. Then, we can take the clock values at the time when the event has been detected and compare it with each other and adjust clocks accordingly.

Most of the mechanisms for clock synchronization used in practice are based on the time dissemination approach, which is simple to implement and scalable. In the 19th century, a system of time balls and time guns was established to disseminate a reference time. In 1883, the Royal Observatory in Greenwich began to drop a large red ball along a mast on the roof every day at exactly 1 pm. The ball was visible in a large area around the observatory and served as a public time signal. In 1852, this system was extended with additional time balls at remote locations, which were triggered over telegraph lines.

The first wireless time dissemination system was established by the U.S. Naval Observatory in 1904. In the following decades, many other radio stations began broadcasting of time signals [63]. Short-wave services operate in the 2.5–20 MHz range to broadcast regular time markers and time announcements by voice while long-wave time services operate at frequencies between 40 and 162 kHz and are mainly used to disseminate time information to radio clocks or equipment that need access to accurate time. Popular examples of such radio stations are WWVB located at Fort Collins, Colorado or DCF77

²Knowing the distance is often not enough since we also need to know the exact way the signal takes, e.g., if the direct path is obstructed and the radio waves are reflected at obstacles.

at Mainflingen near Frankfurt, Germany.

The Global Positioning System (GPS) is based on 24 satellites roughly 20,000 km above the Earth's surface. Each satellite is equipped with a Caesium atomic clock synchronized to UTC. The current time is contained in the GPS messages periodically broadcasted by the satellites. GPS time is usually within 30 nanoseconds from UTC [63] and is thus a precise time source for various applications.

2.2 The Clock Synchronization Problem

In a distributed system, many tasks require a common notion of time to coordinate certain actions or to order events. Typical examples include fusion of sensor data acquired at different spatial locations, e.g., to track the position of acoustic emissions based on the time of arrival [135, 3]. Furthermore, nodes can coordinate access to a communication medium based on a predefined schedule to increase the throughput by avoiding collisions, such as in time-division multiple access (TDMA) protocols.

By design of a distributed system, nodes are independent and thus lack a common clock source. To overcome this problem, we can equip each node with its own local clock to keep track of time. However, these clocks exhibit an intrinsic clock offset and drift, which impairs the quality of a *logical clock* derived from the local clock values. Therefore, the goal of a *clock synchronization algorithm* is to ensure that the logical clock values are as closely synchronized as possible between any two nodes in the network and at any particular time. Certainly, these objectives can only be accomplished if nodes exchange messages about their clock values on a regular basis. Nevertheless, an algorithm should strive to minimize the size and frequency of messages necessary to maintain synchronization, since bandwidth and energy are limited resources in sensor networks.

Internal vs. External Synchronization

For most sensor network applications it is sufficient that all nodes in the sensor network agree on a common time. *Internal synchronization* does not synchronize nodes to an external time reference such as *Coordinated Universal Time (UTC)* but strives to establish a common time base in the network only. For many applications and protocols, such as TDMA or frequency hopping, it is mandatory to have logical clocks synchronized to a common absolute value and rate while the exact time is not important.

External synchronization is required when an application needs to measure exact time intervals or when it is necessary to relate detected events or sensor measurements to real time. Nodes synchronize their clock to an external time reference (e.g., UTC). The common approach is to equip a few nodes with a reference time source such as a GPS receiver. Although,

in practice it is often sufficient to have the sensor network synchronized internally and translate between the network time and a reference time at a gateway node [150].

Global vs. Local Synchronization

Not all tasks in a distributed system necessitate global synchronization. Many wireless networks are organized in a hierarchical manner, e.g., as a tree rooted at a base station. In such a setting, local synchronization is sufficient for a node to maintain a slotted communication and sleep schedule with its neighbors in the tree only [16]. Moreover, it may be tolerable that the synchronization error increases with the hop distance between nodes, which is known as the *gradient* property [40].

Proactive vs. Reactive Synchronization

Coordinated actuation or scheduling of task requires synchronization of nodes *a priori*, which is achieved by re-synchronizing clocks periodically. However, while this keeps the network synchronized at all time, it raises the power consumption due to frequent message exchange. Elson and Römer [36] proposed a *post-facto* synchronization scheme, where clocks are not synchronized in advance, but the timestamps corresponding to an event observed at different nodes are compared.

2.3 Hardware Clocks

The digital hardware clocks of sensor nodes measure time intervals. For each time interval a counter is increased by one. The width of this counter register is usually 8, 16 or 32 bits, depending on the hardware architecture. Eventually, the counter register of the microcontroller will overflow and the hardware clock starts again to count from zero upwards. The length of the time interval depends on the clock frequency. Usually, a counter overflow will trigger an interrupt, which will execute the corresponding interrupt handler for the corresponding counter. By counting the number of overflows by increasing a variable in software, the width of the hardware clock can be extended arbitrarily. The granularity of the clock, i.e., the minimum size of the time interval for which consecutive clock readings are different, directly depends on the *frequency* of the hardware clock. Consequently, a sensor node with a hardware clock running at a higher speed (e.g., 8 MHz) will provide more fine-grained time readings than a clock running at lower speed (e.g., 32 kHz). However, a clock running at a high frequency will also overflow more frequently. Therefore, the interrupt handler is executed more frequently too, which increases the overall power consumption of the node. To operate real-time clocks (RTC) in low-power systems one usually employs

Platform	MCU	RC Oscillator	Crystal Oscillator
Mica2	ATmega128L	1–8 MHz	32 kHz, 7.37 MHz
IRIS	ATmega1281	7.3–8.1 MHz	32 kHz, 7.37 MHz
TelosB	MSP430F1611	0.1–5 MHz	32 kHz

Table 2.1: Comparison of clock sources for common sensor node platforms.

a 32 kHz quartz crystal as the clock source. The counter register of microcontrollers commonly used in sensor nodes, e.g., the Atmel ATmega family or TI’s MSP430 microcontrollers, can be clocked by various sources to offer full flexibility for different application scenarios. Table 2.1 summarizes the clock sources for common sensor node platforms. In general, we can distinguish between internal and external clock sources. A voltage-controlled oscillator (VCO) built into the microcontroller provides a clock source without requiring additional external components. The disadvantage of using a VCO as the hardware clock source is reflected in the limited accuracy that this approach is able to provide, e.g., the 8 MHz internal RC oscillator of the ATmega128L microcontroller varies between 7.6 MHz (-20 °C) and 7.4 MHz (40 °C)³. However, user calibration at runtime allows to achieve $\pm 1\%$ accuracy independent of voltage and temperature conditions.

2.3.1 Clock Drift

If the requirements for the clock stability are more stringent, one relies on external clock sources. A common approach is to use an external quartz crystal to drive an oscillator loop. The frequency deviation of a quartz crystal employed as the oscillating element depends on the quality of the quartz and, hence, on its price. High quality crystal oscillators vary by about 2.5 ppm⁴ over a temperature range from 0 °C to 50 °C but are more costly. Since the unit price for sensor node hardware should be low, sensor nodes often include a cheap crystal oscillator exhibiting drift up to 40 ppm. The frequency stability of a crystal oscillator can be characterized by its long-term and short-term stability, as summarized in Table 2.2. The oscillator frequency may change gradually over days or years due to the physical aging process. During very short time intervals (seconds), one may see small fluctuations from the nominal quartz frequency (noise). Temperature variations can have a large influence on the crystal frequency resulting in large variations in hardware clock drift.

³ATmega128/L Datasheet - <http://www.atmel.com/atmel/acrobat/doc2467.pdf>

⁴ppm = parts per million. An oscillator with 100 ppm running at 1 MHz drifts apart 100 μ s after one second.

Drift Type	Time	Magnitude
Noise (short-term)	seconds	1.0×10^{-9}
Temperature	minutes	2.5×10^{-6}
Aging (long-term)	months	3.0×10^{-7}

Table 2.2: Overview of different factors influencing the clock drift of a crystal oscillator at room temperature [1].

Temperature Effects and Compensation

We measured the influence of the temperature on the hardware clock frequency for the Mica2 sensor node. Two different nodes were placed inside a climate chamber at our laboratory. The ATmega128 microcontroller was configured to toggle an output pin at the hardware clock frequency. The hardware clock signal was connected to an Agilent A53131 high-resolution counter device. Each node is equipped with a Sensirion SHT75 humidity and temperature sensor, which offers precise temperature readings. We gradually changed the temperature in the climate chamber between $-20\text{ }^{\circ}\text{C}$ and $+40\text{ }^{\circ}\text{C}$ over multiple time intervals of 90 minutes duration each.

Figure 2.1 shows the variances in the hardware clock frequency over three measurement cycles. The measurements results clearly show that the frequency of the crystal oscillator is affected by the temperature variations. An increase in the ambient temperature from $0\text{ }^{\circ}\text{C}$ to $20\text{ }^{\circ}\text{C}$ decreases the clock frequency from 921.814 kHz to 921.812 kHz (Node 1) and from 921.813 kHz to 921.810 kHz (Node 2), respectively. Based on these measurement results, we believe that the influence of temperature variations can be reduced, improving the accuracy of clock synchronization when using large beacon intervals.

Temperature compensated crystal oscillators (TCXO) have additional circuitry to compensate for temperature variations. While a TCXO can reduce the temperature effects significantly, this comes at an increased form factor and cost. Therefore, TCXOs have not yet gained widespread adoption on sensor network platforms [126]. An alternative approach is to use the integrated temperature sensor on the sensor node to measure ambient temperature and to adapt the clock frequency accordingly. The temperature-compensated time synchronization (TCTS) algorithm [128] measures the frequency error of the local clock relative to a reference clock with a stable frequency source (e.g., a GPS receiver) and stores it together with the current environmental temperature reading. After the calibration phase has passed, the node continuously measures the temperature and updates its local frequency correction factor.

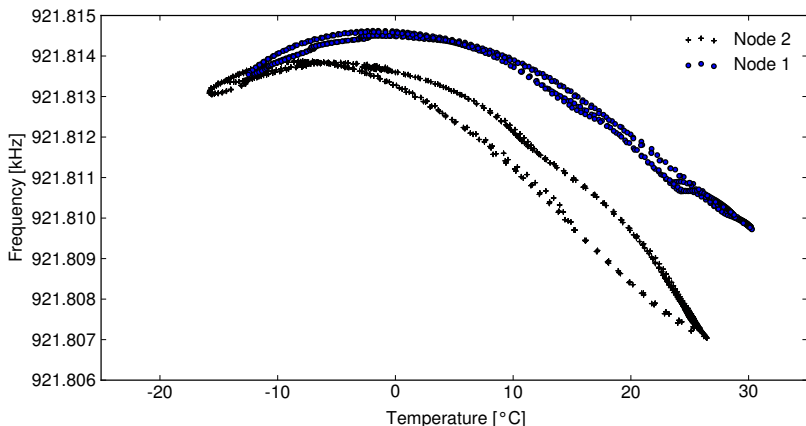


Figure 2.1: Hardware clock frequency of two Mica2 sensor nodes at different temperatures.

2.4 Message Delay

Clock synchronization protocols rely on the exchange of messages over the wireless channel. The current timestamp is read from the local clock and embedded into the payload of a radio packet for transmission. Clearly, the delay of a message exchange cannot be calculated directly from the embedded timestamps. Exchanging the current timestamp of a node by a broadcast introduces errors with magnitudes larger than the required precision due to non-determinism in the message delay. The time it takes from the point of time where the message is passed to the communication stack until it reaches the application layer on a neighboring node is highly non-deterministic due to various sources of errors induced in the message path. The following four components contribute to the message delay in the context of wireless sensor networks [66, 35, 46]:

- The **Send Time** spans the time interval from when the timestamp is written into the payload of the message at the transmitting node until it is passed to the lowest layer of the communication stack. It is not deterministic since other tasks could be executed meanwhile or other messages have to be processed first by the radio driver.
- The **Access Time** spans the waiting time until the radio channel is and the transmission of the message can be started. This depends on the specific medium access protocol (e.g., TDMA).

- The **Propagation Time** accounts for the time needed by the message to propagate from the sender to the receiver. It is usually very small for wireless sensor networks since nodes have only a limited transmission range, e.g., a distance of 300 m corresponds to a propagation time of 1 μ s.
- The **Receive Time** denotes the time between the reception of the message at the receiving node's radio transceiver until the corresponding timestamp is recorded.

A large fraction of the probabilistic error in the message delay can be eliminated by a cross-layer optimization approach for time synchronization protocols. The send, access and receive times can be eliminated from the critical path by time-stamping messages at the MAC layer [91]. The current timestamp is written into the message payload right before the packet is transmitted over the air. Accordingly, at the receiver the timestamp is recorded right after the preamble bytes of an incoming message have been received.

2.4.1 Byte-oriented Radio Transceivers

Byte-oriented radio chips, e.g., the CC1000 chip used on the Mica2 platform, generate an interrupt when a complete data byte has been received and written into the input buffer. The interrupt handler reads the current timestamp from the hardware clock and stores it in the metadata of the message. However, there exists some jitter in the reaction time of the interrupt handler for incoming radio data bytes, depending whether the microcontroller needs to wake up from a low-power mode.

The concurrency model of TinyOS requires that asynchronous access to shared variables has to be protected by the use of `atomic` sections [81]. An interrupt signaled during this period is delayed until the end of the atomic block and the timestamp is taken after that. To achieve clock synchronization with accuracy in the order of a few microseconds, it is mandatory to cope with such cases in order to reduce the variance in the message delay. Therefore, each message is time-stamped multiple times both at the sender and the receiver.

The radio chip generates an interrupt at time b_i when a new data byte has arrived or is ready to be transmitted. The interrupt handler is invoked and reads the current hardware clock value at time t_i as shown in Figure 2.2. The time it takes the radio chip to transmit a single byte over the air is denoted by the `BYTE_TIME`. This constant can be calculated directly from the baud rate and encoding settings of the radio chip. Due to the fact that it takes `BYTE_TIME` to transmit a single byte, the following equation holds for all timestamps:

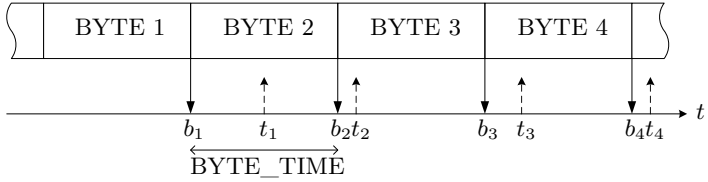


Figure 2.2: Timestamping at the MAC Layer: An interrupt (solid arrow) is generated if a complete byte is received by the CC1000 radio chip. Dashed arrows indicate the time when the interrupt handler takes the timestamp for the current byte.

$$b_{i-1} \leq t_i - \text{BYTE_TIME}$$

Using multiple timestamps, it is hence possible to compensate for the interrupt latency. A better estimation for the timestamp of the i -th byte can be calculated as follows:

$$t'_i = \min(t_i, t'_{i+1} - \text{BYTE_TIME})$$

The timestamps of the first six bytes are used to estimate the arrival time of a packet. A single timestamp for this packet is calculated by taking the average of these timestamps.

2.4.2 Packet-oriented Radio Transceivers

Packet-oriented radio chips like the CC2420 (MicaZ or TelosB) or the RF230 (IRIS mote) unburden the microcontroller from handling every byte transmission separately. Instead, a single interrupt is generated when the start frame delimiter (SFD) has been received, as illustrated in Figure 2.3. Subsequent bytes of the payload are written directly into the FIFO buffer of the receiver. Therefore, compensating jitter in the interrupt handling time is not possible with packet-oriented radio chips.

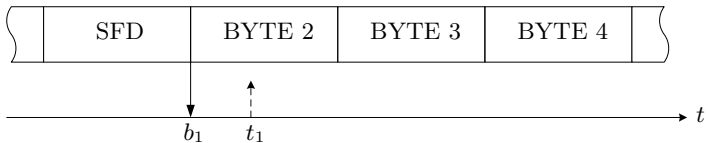


Figure 2.3: Timestamping at the MAC Layer: Packet oriented radio chips like the CC2420 generate a single interrupt when the start frame delimiter (SFD) has been received.

2.4.3 Comparison of Timestamping Accuracy

We used two nodes, one sender and one receiver, to evaluate the accuracy of packet timestamps for different node platforms. The TinyOS implementation of the corresponding radio driver is modified in order to toggle an output pin when a packet is transmitted by the sender and upon packet reception at the receiver. We connected the output pins to an Agilent 53131A counter device to measure the time difference between the rising edge of the two signals. The receiver transmits a new packet every 250 ms and we run the experiment for multiple hours. Both nodes are placed in close proximity to guarantee a high packet reception rate. In the following, we will discuss the measurement results for four different node platforms: MicaZ, TinyNode, TelosB, and ATmega128RFA1. Figure 2.4 depicts the histogram of the message delay for these node platforms, while the measurement results are summarized in Table 2.3.

Mica2

The Mica2 platform is equipped with a Texas Instruments CC1000 radio. A preamble with a size of at least 16 bits is necessary to lock the receiver on an incoming packet. The bytes of the packet are provided by the CC1000 at the received rate over a 3-wire interface. The ATmega128 microcontroller has a single byte input buffer for incoming data from the radio, which triggers an interrupt when the buffer is full. Incoming bytes are timestamped in the interrupt handler by reading the current hardware clock value by software. Since the CC1000 provides a continuous bit stream of the received signal, the last bit of the preamble may not be aligned with the last bit of the microcontroller's input buffer. Therefore, the end of the preamble will be detected only after the next incoming byte has been read, which increases the message delay significantly. However, the additional delay introduced by this mechanism is a multiple of the bit transmission time, which allows to compensate for it by the microcontroller. We observe a message delay of 605.906 μs on average with a standard deviation of 2.738 μs .

TinyNode

The TinyNode platform [29] features a Texas Instrument MSP430F1611 microcontroller and a Semtech XE1205 radio transceiver (868 Mhz, FSK modulation). The X1205 radio has an internal 16 Byte FIFO buffer and an automatic pattern recognition mechanism to detect the packet preamble. An interrupt is triggered on a pattern match and the corresponding timestamp is recorded in the interrupt handler. We measured an average message delay of 825.834 μs and a standard deviation of 0.247 μs .

Platform	MCU	Radio	Message Delay [μs]	
			Mean	Std. dev.
Mica2	ATmega128L	CC1000	605.906	2.738
TinyNode	MSP430F1611	XE1205	825.834	0.247
TelosB	MSP430F1611	CC2420	3.158	0.040
ATmega128RFA1	ATmega1281	RF231	217.702	0.415

Table 2.3: Message delay for different node platforms and radio chips.

TelosB

The Berkeley TelosB platform [112] is built around the same MPS430 microcontroller as on the TinyNode platform, but it uses a Texas Instruments CC2420 radio (2.4 GHz, DSSS-O-QPSK modulation). The TelosB platform provides hardware timestamping support by connecting a special output pin of the CC2420 radio to a timer capture input of the microcontroller. The radio sets this output pin to high as soon as the start-of-frame delimiter (SFD) in the packet header has been transmitted or received. Therefore, timestamps of incoming packets can be recorded by the hardware and are not affected by the jitter in the latency of the interrupt handler. As expected, this results in a small average message delay of 3.158 μs with a standard deviation of only 0.040 μs . These values are in accordance with the measurement results reported by Schmid et al. [126].

ATmega128RFA1

The Atmel ATmega128RFA1 combines a low-power microcontroller and a 2.4 GHz radio on a single chip. This allows the microcontroller to operate the radio transceiver through direct frame buffer and register access. Furthermore, eight different interrupt events can be configured such as an interrupt indicating the start of a packet reception. The integrated MAC symbol counter allows to timestamp the SFD of an incoming packet automatically. It can be operated from the integrated real-time clock (32.768 kHz) or a 16 MHz crystal. However, we measured the message delay of the ATmega128RFA1 by toggling a pin on the sender right after writing the transmit command to the corresponding register. At the receiver, a pin is toggled immediately after the execution of the interrupt handler for a packet reception has started. Our measurements show an average message delay of 217.702 μs and a standard deviation of 0.415 μs . Clearly, it is expected that the message delay is decreased when timestamping the incoming packet using the MAC symbol counter.

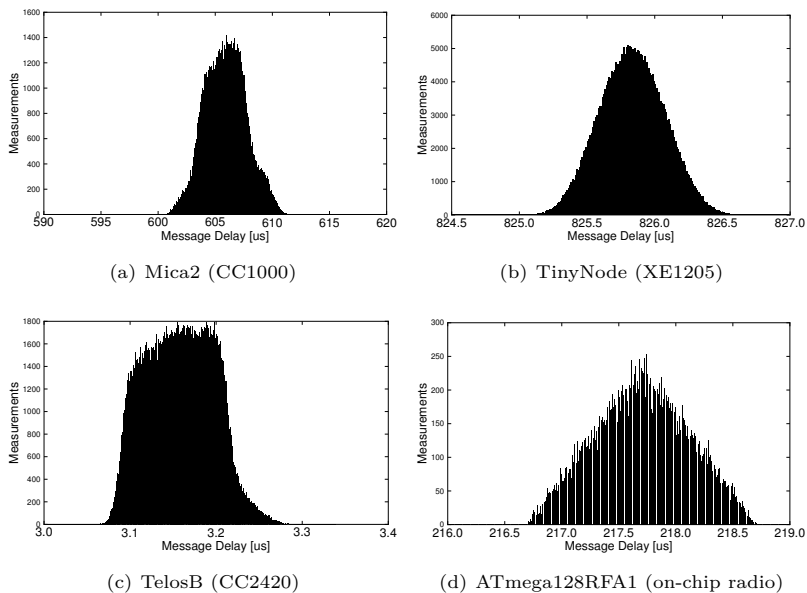


Figure 2.4: Measurement results for the message delay of different sensor node platforms and radio chips.

2.5 Related Work

Clock synchronization has been studied extensively ever since the advent of distributed systems, later also specifically in the context of wireless sensor networks. In this section, we give an overview of the most important algorithmic ideas and practical implementations of synchronization protocols.

2.5.1 Distributed Systems

A distributed system is a collection of different processes that communicate by exchanging messages. Due to the spatial separation of two processes, the message will not reach the receiver instantaneously but only after a certain delay. Lamport proposed a simple mechanism to order events in a distributed system [74]. Each process maintains a logical clock, realized as a counter, which is incremented between any two successive events. Furthermore, the current clock value of a process is appended to each outgoing message. Upon receiving a message, the logical clock of the receiver is incremented so that it is greater than the maximum of the previous clock value and the received

clock value. This mechanism allows to order events generated at different processes, i.e., to say which event happened before another event.

The fundamental problem of clock synchronization has been studied extensively and many theoretical results have been published which give bounds for the clock synchronization error, denoted as *clock skew*, and communication costs [90, 106]. Biaz and Welch [11] proved a worst-case lower bound of $\Omega(D)$ on the network-wide synchronization error, where D denotes the network diameter. Given the hardware clock drift, the algorithm presented in [139] minimizes the global skew.

The *Network Time Protocol (NTP)* [95], is the standard Internet protocol for time synchronization. It builds a hierarchical structure of time servers, some of them synchronized to atomic clocks or GPS receivers, to distribute accurate time for Internet hosts. Since synchronization messages travel over different paths, characterized by varying loads and delays, NTP employs weighted averaging and a phase-locked loop to account for large variations in the delay of synchronization messages. The accuracy provided by NTP is in the order of a few milliseconds.

Gradient Clock Synchronization

The *gradient clock synchronization problem* was first introduced by Fan and Lynch in [40]. The gradient property of a clock synchronization algorithm requires that the clock skew between any two nodes is bounded by the distance (uncertainty in the message delay) between the two nodes. They prove a lower bound for the clock skew of $\Omega(d + \frac{\log D}{\log \log D})$ for two nodes with distance d , where D is the network diameter. This lower bound also holds if delay uncertainties are neglected and an adversary can decide when a sync message will be sent [94]. Lenzen et al. [79] proposed a distributed clock synchronization algorithm guaranteeing clock skew $\mathcal{O}(\log D)$ between neighboring nodes while the global skew between any two nodes is bounded by $\mathcal{O}(D)$. However, we argue that these well-established theoretical lower bounds are based on too harsh assumptions to describe the practical limitations of clock synchronization in real-world systems. Typically they are proved by taking adversarial control of message delay jitters and/or clock drifts. However, jitters are random in nature and clock drift does not vary arbitrarily, at least not arbitrarily quickly.

2.5.2 Wireless Sensor Networks

Traditional approaches such as NTP are not directly adaptable for sensor network applications [36]. First of all, their need to exchange messages at a high rate to determine an accurate estimation of clock drift and offset cannot be reconciled with the prevailing energy constraints in sensor networks. Secondly, protocols such as NTP provide synchronization accuracy in the order

of milliseconds only, which might not be sufficient for certain coordination tasks such as media access, or acoustic localization. For these reasons, clock synchronization problem has been reconsidered when the need for accurate time emerged for sensor network applications. In the following, we summarize hardware and software based approaches for clock synchronization in wireless embedded systems.

Peripheral Clock Receivers

In wireless networks, the classic solution is to integrate passive receivers into nodes to decode time broadcasts from other sources such as GPS, long-wave radio stations [21] or FM radio [83]. However, limitations in terms of cost, form factor and power consumption in general render it infeasible to equip each sensor node with additional hardware. Moreover, line of sight to the GPS satellites is needed, restricting the use of GPS to outdoor applications. Hybrid approaches integrate sophisticated clock sources into a few nodes only, which disseminate the reference time into the sensor network.

Rowe et al. [120] present a *Syntonistor* device, which detects changes in the magnetic field in the proximity of AC power lines and generates a stable clock source from it. This allows to synchronize the rate of the clocks to the 50/60 Hz frequency of the power line, but cannot eliminate the clock offset between nodes. Consequently, a network flooding is necessary to provide nodes a synchronization point between the reference time and the local clock.

Algorithms for Clock Synchronization

Sensor networks require sophisticated algorithms for clock synchronization since the hardware clocks in sensor nodes are often simple and may experience significant drift. Moreover, the multi-hop character of wireless sensor networks complicates the problem, as one cannot simply employ a standard client/server clock synchronization algorithm as in wired networks.

As research in sensor networks evolved during the last years, many different approaches for time synchronization were proposed. Römer presents a reactive synchronization protocol where events are time-stamped with the local clock [118]. When such a timestamp is passed to another node, it is converted to the local timestamp of the receiving node.

Receiver-Receiver Synchronization

Reference Broadcast Synchronization (RBS) [35] exploits the broadcast nature of the physical channel to synchronize a set of receivers with one another. A reference node is elected within each cluster to synchronize all other nodes. Since differences in the propagation times can generally be neglected in sensor networks, a reference message eventually arrives at the same instant at all receivers. The timestamp of the reception of a broadcast message is recorded

at each node and exchanged with other nodes to calculate relative clock offsets. Although being designed for single-hop time synchronization only, RBS can be utilized for multi-hop communication as component of other clock synchronization algorithms [68].

Sender-Receiver Synchronization

The *Timing-sync Protocol for Sensor Networks (TPSN)* [46] aims to provide network-wide time synchronization. The TPSN algorithm elects a root node and builds a spanning tree of the network during the initial level discovery phase. Each node is assigned a fixed level. In the synchronization phase of the algorithm, nodes synchronize to their parent in the tree by a two-way message exchange. Using the timestamps embedded in the synchronization messages, the child node is able to calculate the transmission delay and the relative clock offset. MAC layer time-stamping is used to reduce possible sources of uncertainty in the message delay. However, TPSN does not compensate for clock drift which makes frequent re-synchronization mandatory. In addition, TPSN causes a high communication overhead since a two-way message exchange is required for each child node.

The *Lightweight Time Synchronization (LTS)* [147] protocol has two mechanisms to synchronize nodes to a reference node in the network. The centralized approach is based on a spanning tree constructed by network flooding. The reference node initiates the pair-wise synchronization along the edges in the tree. The interval between successive resynchronizations is based on the maximum depth of the spanning tree to meet the desired quality of synchronization. Alternatively, the protocol can also operate in a distributed variant, where pair-wise synchronization along the path to a reference node can be initiated by any node.

Both *Tiny-Sync* and *Mini-Sync* [161] employ a two-way message exchange to collect data points for drift and offset estimates. Using the assumption that clock drift is relatively stable over a short time interval and by employing a line-fitting approach, a minimum and maximum bound on the drift and offset is calculated satisfying the constraints given by the data points. Finally, the relative drift and offset estimation is calculated based on the average value for the slope and the axis intercept. Tiny-Sync keeps only those four data points that result in the best bounds on the drift and offset estimation. Although this method is efficient in terms of memory consumption, it may not always calculate the optimal estimation. In contrast, Mini-Sync stores a larger number of data points and will only remove a data point if it is irrelevant for the calculation of the optimal solution. Interestingly, the performance of the two algorithms is very similar in practice.

Flooding-based Protocols

Two-way synchronization methods require a separate message exchange between a parent and each child. These shortcomings are tackled by the *Flooding-Time Synchronization Protocol (FTSP)* [91]. A root node is elected which periodically floods its current timestamp into the network forming an ad-hoc tree structure. MAC layer time-stamping reduces possible sources of uncertainty in the message delay. Each node uses a linear regression table to convert between the local hardware clock and the clock of the reference node. The root node is dynamically elected by the network based on the smallest node identifier. After initialization, a node waits for a few rounds and listens for synchronization beacons from other nodes. Each node sufficiently synchronized to the root node starts broadcasting its estimation of the global clock. If a node does not receive synchronization messages during a certain period, it will declare itself the new root node. FTSP is used as the clock synchronization protocol for many sensor network deployments. Recently, the original FTSP protocol has been extended to make it more robust against temperature changes [129] and clock rate variations [127].

The *Routing Integrated Time Synchronization protocol (RITS)* [121] uses a reactive approach to provide post-facto synchronization. Detected events are time-stamped with the local time and reported to the sink. When such an event timestamp is forwarded towards the sink node, it is converted from the local time of the sender to the receiver's local time at each hop. A drift compensation strategy improves the accuracy of this approach.

Rapid Time Synchronization (RATS) [71] employs a network-wide broadcast to disseminate the local time of the root node. As in the FTSP protocol, linear regression is used to estimate and compensate clock drifts.

Non-hierarchical Protocols

The *Reachback Firefly Algorithm (RFA)* is a fully distributed synchronization algorithm inspired from the way neurons and fireflies spontaneously synchronize [152]. Each node periodically generates a pulse (message) and observes pulses from other nodes to adjust its own firing phase. The authors report that a synchronization accuracy of 100 μ s can be achieved with this approach. RFA only provides *synchronicity*, nodes agree on the firing phases but do not have a common notion of time. Another shortcoming of RFA is the fact that it has a high communication overhead. Li et al. [84] proposed an asynchronous time diffusion algorithm where nodes adjust their clock value based on the value of its neighbors by averaging. Another approach to exploit all communication links has been proposed in [136], where the algorithm seeks to minimize the sum of the squared errors over all edges. This can be accomplished in a distributed manner by applying gradient descent on the corresponding linear system.

3

Gradient Clock Synchronization

Although multi-hop clock synchronization has been studied extensively in the last decade, we believe that there are still facets that are not understood well, and eventually need to be addressed. One such issue is *locality*: Naturally, one objective in clock synchronization is to minimize the clock error, also called *skew*, between any two nodes in the network, regardless of the “distance” between them. This is known as *global* clock skew minimization. Indisputable, having two far-away nodes well-synchronized is a noble goal, but is it really what is required most? In this chapter, we argue that accurate clock synchronization between neighboring nodes is often at least as important. In fact, all examples mentioned earlier tolerate a suboptimal global clock synchronization: Guessing the location of a commonly sensed acoustic signal needs a precise clock synchronization between all the nodes that are able to sense the signal. Similarly, in a MAC layer that is optimized for throughput or energy, we care that possibly interfering (neighboring) nodes have a precise clock. In contrast, global skew is not of great concern, it is perfectly tolerable if far-away nodes have larger pair-wise errors. This is known as *local* clock skew minimization. Optimally, we would like to have a clock synchronization protocol that is precise in the direct neighborhood, and maybe a bit less so in the more extended neighborhood.

In the theory community, clock synchronization has been studied for many years, recently with a focus on the local (also known as *gradient*) clock skew, e.g., [40] and [87]. The aim of this chapter is to investigate whether these

theoretical insights carry over to practice. In particular, we will propose the Gradient Time Synchronization Protocol (GTSP), a clock synchronization protocol that excels primarily at local clock synchronization. It is inspired by a long list of theoretical papers, originating in the distributed computing community [74, 90, 139, 106], lately also being adopted for wireless sensor networks [84, 125]. As such, GTSP is completely distributed, relying only on local information, requiring no reference node or tree construction. We argue that this approach results in a better average synchronization between neighbors while still maintaining a tolerable global skew. A thorough evaluation of our algorithm is performed on different testbeds and by simulations. The work presented in this chapter is based on [138].

3.1 System Model

In this section, we introduce the system model used throughout the rest of this thesis. We assume a network consisting of a number of nodes equipped with a hardware clock subject to clock drift. Furthermore, nodes can convert the current hardware clock reading into a logical clock value and vice versa.

We model a (sensor) network of $|V| =: n$ nodes as undirected graph $G = (V, E)$ of diameter D , where edges $\{v, w\} \in E$ represent bidirectional communication links. A message sent by a node $v \in V$ is received by all of its neighbors $w \in N_v$. For the purpose of our analysis only, we assume that communication is reliable and the network is static. However, it takes messages some time to arrive at their destination. More precisely, the *message delay* is the time between a node deciding to send a message and the receiver finally processing it. Even if we take into account the deterministic delays, the system still suffers from random deviations of the message delay, the so called *message delay jitter* (see Section 2.4).

3.1.1 Hardware Clock

Each sensor node v is equipped with a hardware clock $H_v(t)$. The clock value at time t is defined as

$$H_v(t) = H_v(0) + \left[\int_0^t h_v(\tau) d\tau \right]$$

where $H_v(0)$ is the *hardware offset* and $h_v(\tau)$ is the *hardware clock rate* of node v at time τ .

The rate of the hardware clock depends on conditions like the ambient temperature, the supply voltage, or crystal quartz aging, all changing over time, but even under perfectly stable and identical environmental conditions the clocks of different nodes will run at different speeds. However, we assume that

$$\forall v \in V \forall t : |1 - h_v(t)| \leq \rho \ll 1,$$

i.e., the clocks exhibit a bounded drift. Typical hardware clock drifts are around 40 ppm, while the fluctuations due to temperature are substantially smaller (see Section 2.3.1).

3.1.2 Logical Clock

Since other hardware components may depend on a continuously running hardware clock, its value should not be adjusted manually. Instead, a logical clock value $L_v(\cdot)$ is computed as a function of the current hardware clock. The logical clock value $L_v(t)$ represents the synchronized time of node v . It is calculated as follows:

$$L_v(t) = \int_{t_0}^t h_v(\tau) \cdot l_v(\tau) d\tau + \theta_v(t_0)$$

where $l_v(\tau)$ is the *relative logical clock rate* and $\theta_v(t_0)$ is the clock offset between the hardware clock and the logical clock at the reference time t_0 . The logical clock is maintained as a software function and is only calculated on request based on a given hardware clock reading.

3.1.3 Synchronization Error

The goal of any clock synchronization algorithm is to ensure that these values are as closely synchronized as possible both between any two nodes in the network. The difference in the logical clock value $L(t)$ between nodes at a specific time t determines the synchronization error. We thereby differentiate between the (maximum) *network synchronization error (global skew)* defined as

$$\mathcal{G}(t) := \max_{v,w \in V} \{|L_v(t) - L_w(t)|\}$$

and the (maximum) *neighbor synchronization error (local skew)*

$$\mathcal{L}(t) := \max_{v \in V, w \in N_v} \{|L_v(t) - L_w(t)|\}.$$

Correspondingly, the average global and local skews are the average skews between all pairs of nodes, respectively neighbors.

3.1.4 Message Complexity

Synchronizing the clocks of different sensor nodes involves the exchange of timing information over the radio channel. Given the memory and energy constraints of embedded nodes, communication is only possible infrequently and at a low data rate. Therefore, we assume that a single message can contain a payload of a few bytes only, i.e., the typical payload size of a TinyOS packet is 28 bytes, and that the exchange of time information should be kept at a low frequency, i.e., one synchronization beacon sent per node within the a period of 30 seconds.

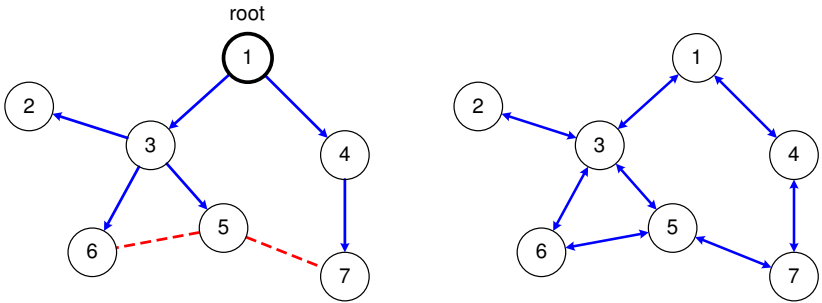


Figure 3.1: The graph on the left represents the topology used by a tree-based synchronization protocol with node 1 as reference clock (root), where every node in the tree synchronizes with its parent; in this example we would expect nodes 5 and 7 to synchronize in a less than optimal manner even though they are direct neighbors, because they are part of different subtrees. On the right, we see the idea of a gradient synchronization protocol: Every node synchronizes with all its neighbors in the communication graph. No root node is necessary.

3.2 Hierarchical Clock Synchronization

State-of-the-art multi-hop clock synchronization protocols such as FTSP [91] are designed to optimize the global skew. However, as we will show in this chapter, there is room for improvement regarding the local skew. This is not really surprising, as FTSP and similar protocols work on a spanning tree, synchronizing nodes in the tree with their parents, and ultimately with the root of the tree. Neighboring nodes that are not closely related in the tree, i.e., where the closest common ancestor even is the root of the tree, will not be synchronized well because errors propagate down differently on different paths of the tree, see Figure 3.1. Eliminating all the deterministic sources of errors, the remaining two-hop error would be totally symmetric in the best case [137]. Indeed, every hop will experience some kind of inevitable random error δ . As randomly distributed errors sum up according to the square-root function on each hop, the expected error between head and tail of a chain of k nodes is in the order of $\delta\sqrt{k}$. Therefore, any two nodes can be expected to experience an error in the order of the square-root of their distance in the tree.

We can show that the stretch of the spanning tree is at least \sqrt{n} in every grid of n nodes. In other words, there exist two neighboring nodes that are distance \sqrt{n} apart in the spanning tree of the grid. Using the example topology depicted in Figure 3.2, we choose an arbitrary Node A as the root of the spanning tree. Starting from the root node, we then walk between the

In a network consisting of sensor nodes with perfectly calibrated clocks, i.e., assuming no drift, time progresses at the same rate throughout the network. It remains to calculate once the relative offsets amongst the nodes, so that they agree on a common global time. However, real hardware clocks exhibit relative drift in the order of up to 80 ppm leading to a continually increasing synchronization error between nodes.

Therefore, it is mandatory to repeat the synchronization process frequently to guarantee certain bounds for the synchronization error. However, precisely synchronized clocks between two synchronization points can only be achieved if the relative clock drift between nodes is compensated. In structured clock synchronization algorithms all nodes adapt the rate of their logical clock to the hardware clock rate of the reference node. This approach requires that a root node is elected and a tree structure of the network is established. Synchronization algorithms operating on structured networks have to cope with topology changes due to link failures or node mobility.

In a clock synchronization algorithm that should be completely distributed and reliable to link and node failures, it is not practicable to synchronize to the clock of a reference node. Therefore, our clock synchronization algorithm strives to agree with its neighbors on the current logical time. Having synchronized clocks is a twofold approach, one has to agree both on a common logical clock rate and on the absolute value of the logical clock.

3.3.1 Drift Compensation

We define the *absolute logical clock rate* $x_v(t)$ of node v at time t as follows:

$$x_v(t) = h_v(t) \cdot l_v(t)$$

Each node v periodically broadcasts a synchronization beacon containing its current logical time $L_v(t)$ and the relative logical clock rate $l_v(t)$. Having received beacons from all neighboring nodes during a synchronization period, node v uses this information to update its absolute logical clock rate as follows:

$$x_v(t_{k+1}) = \frac{\left(\sum_{u \in \mathcal{N}_v} x_u(t_k)\right) + x_v(t_k)}{|\mathcal{N}_v| + 1} \quad (3.1)$$

where \mathcal{N}_v is the set of neighbors of node v .

It is important to note that in practice node v is unable to adjust x_v itself since it has no possibility to measure its own hardware clock rate h_v . Instead, it can only update its relative logical clock rate $l_v = \frac{x_v}{h_v}$ as follows:

$$l_v(t_{k+1}) = \frac{\left(\sum_{u \in \mathcal{N}_v} \frac{x_u(t_k)}{h_u(t_k)}\right) + l_v(t_k)}{|\mathcal{N}_v| + 1} \quad (3.2)$$

We have to show that by using this update mechanism all nodes converge to a common logical clock rate x_{ss} , which means that:

$$\lim_{t \rightarrow \infty} x_v(t) = \lim_{t \rightarrow \infty} h_v(t) \cdot l_v(t) = x_{ss}, \forall v$$

We assume that the network is represented as a graph $G(V, E)$ with the nodes as vertices and edges between nodes indicating a communication link between the two nodes. For the purpose of our analysis, we assume that the network is static and fully connected, i.e., there exists a path between any two nodes. Using matrix multiplication the update of the logical clock rates performed in Equation 3.1 can be written as:

$$x[k+1] = A[k] \cdot x[k]$$

where the vector $x = (x_1, x_2, \dots, x_n)^T$ contains the logical clock rates of the nodes. Thereby, it is assumed that the clock rates are updated simultaneously at all nodes. The entries of the $n \times n$ matrix A are defined in the following way:

$$a_{vu} = \begin{cases} \frac{1}{|\mathcal{N}_v|+1} & \{v, u\} \in E \\ 0 & \text{otherwise} \end{cases}$$

where $|\mathcal{N}_v|$ is the degree of node v . Since all rows of matrix A sum up to exactly 1, it is *row stochastic*. Initially, the logical clock of each node v has the same rate as the hardware clock ($x_v(0) = h_v(0)$) since the logical clock is initialized with $l_v(0) = 1$. It can be shown that all the logical clock rates will converge to a steady-state value x_{ss} :

$$\lim_{k \rightarrow \infty} x[k] = x_{ss} \mathbf{1} \tag{3.3}$$

The convergence of Equation 3.3 depends on whether the product $\prod_{t=0}^{\infty} A(t)$ of non-negative stochastic matrices has a limit. It is well-known that the products of row stochastic matrices converge if the graph corresponding to the matrices $A(t)$ is strongly connected [157, 18].

3.3.2 Offset Compensation

Besides having all nodes agree on the rate at which the logical clock is advanced, it is also necessary to synchronize the actual clock values themselves. Again, the nodes have to agree on a common clock value, which can be obtained by calculating the average of the clock values as for the drift compensation. A node i updates its logical clock offset θ_i as follows:

$$\theta_i(t_{k+1}) = \theta_i(t_k) + \frac{\sum_{j \in \mathcal{N}_i} L_j(t_k) - L_i(t_k)}{|\mathcal{N}_i| + 1} \tag{3.4}$$

However, using the average of all neighbors as the new clock value is problematic if the offsets are large. During node startup, the hardware clock register is initialized to zero, resulting possibly in a huge offset to nodes that are already synchronized with the network. Such a huge offset would force all other nodes to turn back their clocks which violates the causality principle. Instead, if a node learns that a neighbor's clock is further ahead than a certain threshold value, it jumps to the neighbors clock value.

By employing this bootstrap mechanism, a node joining the network gets synchronized quickly with the rest of the network. In the worst case it can take up to $\mathcal{O}(D)$ time to have all nodes loosely synchronized, where D is the diameter of the network. Since the logical clock rate of a node that recently joined the network may not be synchronized with the network yet, its clock value will start to drift apart immediately after the initial synchronization point. The resulting synchronization error is bounded by the hardware clock drift accumulated during a synchronization interval.

3.3.3 Computation and Memory Requirements

Computation of the logical clock rate involves floating point operations. Since most sensor platforms support integers only, floating point arithmetic has to be emulated using software libraries. However, since the range of the logical clock rate is bounded by the maximum clock drift, computations can greatly benefit from the use of fixed point arithmetic.

Besides the computational constraints of current sensor hardware, data memory is also very limited and the initial capacity of data structures has to be specified in advance. The synchronization algorithm requires storing information about the relative clock rates of its neighbors which are used in Equation 3.2. Since the capacity of the data structures is limited, the maximal number of neighbors a node accounts for in the calculations is also limited and a node possibly has to discard crucial neighbor information. However, ignoring messages from a specific neighbor does still lead to consensus as long as the resulting graph remains strongly connected. Since the capacity constraints are only a problem in very dense networks, it is very unlikely that a partitioning of the network graph is introduced.

3.3.4 Energy Efficiency

Radio communication consumes a large fraction of the energy budget of a sensor node. While the microcontroller can be put into sleep mode when it is idle, thus reducing the power consumption by a large factor, the radio module still needs to be powered to capture incoming message transmissions. Energy-efficient communication protocols, e.g., B-MAC [111], employ scheduled radio duty-cycling mechanisms to lower the power consumption and

thus prolonging battery lifetime. Since the exact timing when synchronization messages are sent is not important, GTSP can be used together with an energy-efficient communication layer. In addition, a node can estimate the current synchronization error to its neighbors from the incoming beacons in order to dynamically adapt the interval between synchronization beacons. If the network is well synchronized, the beacon rate can be lowered to save energy. The communication overhead of GTSP is comparable with FTSP since both algorithms require each node to broadcast its time information only once during a synchronization period.

3.4 Simulations

We implemented GTSP in the Sinalgo network simulator [44], which allows rapid prototyping of algorithms using Java. Although simulations cannot capture the exact behavior of the hardware (e.g., effects such as interrupt latency, interferences), we gain a first impression how GTSP performs on a large scale network. For the simulation of the sensor nodes, we modeled the hardware clock of a node in software. At the start of a simulation run, each node is initialized with a random hardware clock drift of 40 ppm and a random start value. The variance in the message delay is modeled by a normally distributed random variable with zero mean and a standard deviation of one clock tick. We measured the average synchronization error between neighbors and the average network-wide synchronization error for different network topologies. For each network setting we obtain the mean synchronization errors averaged over 10 simulation runs with different random seed, as depicted in Figure 3.3.

GTSP performs best when the nodes form a grid, which has the smallest network diameter amongst the studied topologies. Not surprisingly, the worst clock accuracy is achieved when the nodes are placed in a line, forming a network with maximal diameter. The simulation results clearly show that synchronization accuracy between neighboring nodes depends on the network topology and the network diameter.

3.5 TinyOS Implementation

This section describes the implementation of our gradient clock synchronization algorithm on the Mica2 sensor nodes in TinyOS.

3.5.1 Target Platform

The hardware platform used for the implementation of the algorithm is the Mica2 sensor node from Crossbow. It features an ATmega128L low-power microcontroller from Atmel with 4 KBytes of RAM, 128 KBytes of program

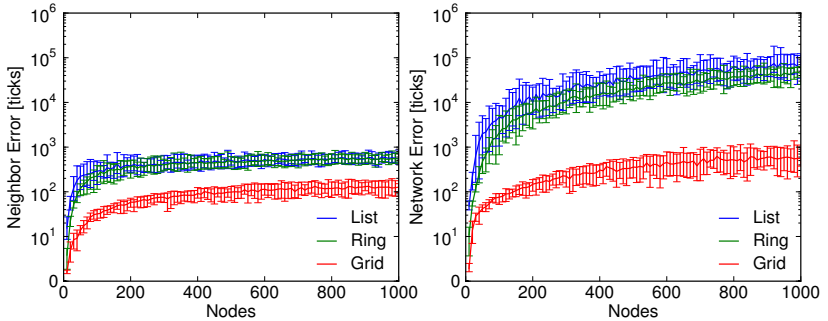


Figure 3.3: Average neighbor (left) and network synchronization errors (right) measured by simulating GTSP on different network topologies. Mean synchronization errors are averaged over ten runs, error bars indicate values of runs with maximum and minimum outcome.

ROM and 512 KBytes of external flash storage. The CC1000 radio module has been designed for low-power applications and offers data rates up to 76.8 kBaud using frequency shift keying (FSK).

The ATmega128L microcontroller has two built-in 8-bit timers and two built-in 16-bit timers. The Mica2 board is equipped with two different quartz oscillators (32 kHz and 7.37 MHz) which can be used as clock sources for the timers. Timer3 is configured to operate at $\frac{1}{8}$ of the oscillator frequency (7.37 MHz) leading to a clock frequency of 921 kHz. Since Timer3 is sourced by an external oscillator, it is also operational when the microcontroller is in low-power mode. We employ Timer3 to provide our system with a free-running 32-bit hardware clock which offers a precision of a microsecond. This approach on the Mica2 node offers better clock granularity as compared to more recent hardware platforms that lack a high frequency external oscillator (see Table 2.1).

3.5.2 Software Architecture

The implementation of GTSP on the Mica2 platform is done in TinyOS 2.1. The protocol implementation provides time synchronization as service for an application running on the node. The architecture of the time synchronization component and its integration with the other system components is shown in Figure 3.4.

The *Synchronization Module* periodically broadcasts a synchronization beacon containing the current logical time $L_v(t)$ and the relative logical clock rate $l_v(t)$. Each node is overhearing messages sent by neighboring nodes. The timestamp contained in the synchronization beacons is used to update the

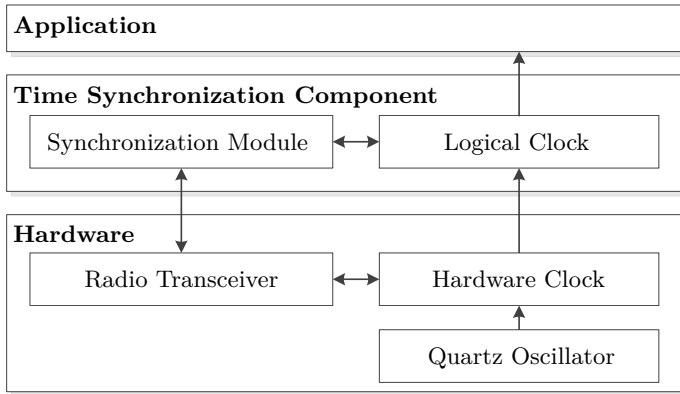


Figure 3.4: Architecture of the time synchronization service and its integration within the hardware and software platform. Arrows indicate the flow of information between different components.

current offset between the hardware and the logical time and the rate of the logical clock according to Equations 3.2 and 3.4. The hardware and logical time when the most recent synchronization beacon of each neighbor has been received is stored in a neighbor table.

By overhearing synchronization beacons a node will learn when another node joins its neighborhood. When no beacon messages were received from a node for several consecutive beacon intervals, the link to this node is assumed to be broken and the node is removed from the neighbor table. The capacity of the neighbor table is limited by the data memory available on the node. An upper bound for the required capacity is the maximum node degree in the network. However, as long as the resulting network graph stays connected it is possible to ignore synchronization beacons from a specific neighbor. The default capacity of the neighbor table in our implementation is set to 16.

Furthermore, the time interval between synchronization beacons can be adapted dynamically. This allows to increase the frequency of beacons during the bootstrap phase or when a new node has recently joined the network. On the other side, if the system is in the steady-state, i.e., all nodes are quite well synchronized to their neighbors, reducing the number of sent beacons can save energy.

The *Logical Clock* module stores the clock offset and relative clock rate of the logical clock. Interface 3.1 sketches the commands provided to other components by the *TimeSync* interface. Other components within the TinyOS application can retrieve the current hardware and logical timestamps and convert timestamps between the different clock domains. Furthermore, the

Interface 3.1: Time synchronization interface declaration in TinyOS

```
interface TimeSync {
    command uint32_t getHardwareTime();
    command uint32_t getLogicalTime();

    command void convertToLogicalTime(uint32_t *time);
    command void convertToHardwareTime(uint32_t *time);

    command bool isSynced();
    command float getClockDrift();
}
```

`isSynced()` command indicates whether the node is currently in synchronization to its neighbors or not.

3.6 Experimental Evaluation

In this section, we evaluate the performance of the Gradient Time Synchronization Protocol (GTSP). Evaluating clock synchronization algorithms can be done by comparing various metrics such as precision, energy consumption, or communication overhead. In the remainder of this chapter, we restrict our evaluation to the precision achieved by the synchronization algorithm.

Measuring the instantaneous error between logical clock of different nodes is only possible at a common time instant, e.g., when all nodes can observe the same event simultaneously. A general practice when evaluating time synchronization algorithms for sensor networks is to transmit a message as a reference broadcast. All nodes are placed in communication range of the reference broadcaster. The broadcast message arrives simultaneously at all nodes (if the minimal differences in the propagation delay are neglected) and is time-stamped with the hardware clock. The corresponding logical clock value is used to calculate the synchronization error to other nodes. Two different metrics are used throughout the evaluation in this thesis: the *Average Neighbor Error* measures the average pair-wise differences in the logical clock values of nodes that are *direct* neighbors in the network graph while the *Average Network Error* is defined as the average pair-wise synchronization error between *arbitrary* nodes.

3.6.1 Small-Scale Testbed Experiments

We evaluated the implementation of GTSP by experiments on a testbed which consists of 20 Mica2 sensor nodes. Experiments with the identical setup are also performed for FTSP which is the standard time synchronization protocol in TinyOS. All nodes are placed in close proximity forming a

single broadcast domain. In addition, a base station node is attached to a PC to log synchronization messages sent by the nodes. To facilitate measurements on different network topologies, a virtual network layer is introduced in the management software of the sensor nodes. Each node can be configured with a whitelist of nodes from which it will further process incoming messages, whereas packets from all other nodes are ignored. Using this virtual network layer different network topologies can be enforced by software.

The base station periodically broadcasts probe messages to query the current logical time of all the nodes. The interval between time probes is uniformly distributed between 18 and 22 seconds. To reduce radio collisions with time synchronization messages, nodes do not reply with the current time value. Instead, the current local timestamp and the estimated logical timestamp are logged to the external flash memory.

Experimental Results for GTSP

At the beginning of the experiment, the configuration parameters for GTSP were set for all nodes. The synchronization algorithm was started on every node at random during the first 30 seconds of the experiment. Synchronization beacons are broadcast every 30 seconds. The offset threshold parameter is set to 10 ticks. Therefore, a node adjusts its logical clock value if the logical clock of a neighbor is further ahead than 10 μs . Right after the initialization all nodes have zero logical clock offset and the rate of the logical clock corresponds to the hardware clock rate. We denote the period between synchronization beacons by P and the network diameter by D . It takes up to $D \cdot P$ time until all nodes raised their logical clock to the value of the node having the highest logical clock value.

After having received the second beacon from a neighboring node, nodes can estimate the rate of the neighbor's logical clock (relative to the local hardware clock). To reduce the effects of jitter in the message delay, the estimated clock rates of the neighbors are filtered by a moving average filter with $\alpha = 0.6$. The experiment lasted for approximately 6 hours which resulted in around 1000 time probes logged to the flash storage of the sensor nodes. The measurement results for GTSP on a ring of 20 Mica2 nodes is depicted in Figure 3.5. It can be seen that GTSP achieves an average synchronization error between neighbors of 2.96 μs after the initialization phase has been completed ($t > 3000$ s). The average network synchronization error is 8.94 μs for the same interval. For our testbed setup consisting of 20 nodes placed in a ring, it takes roughly 30 minutes until the algorithm converges, which is comparable to the convergence time of FTSP, see next subsection.

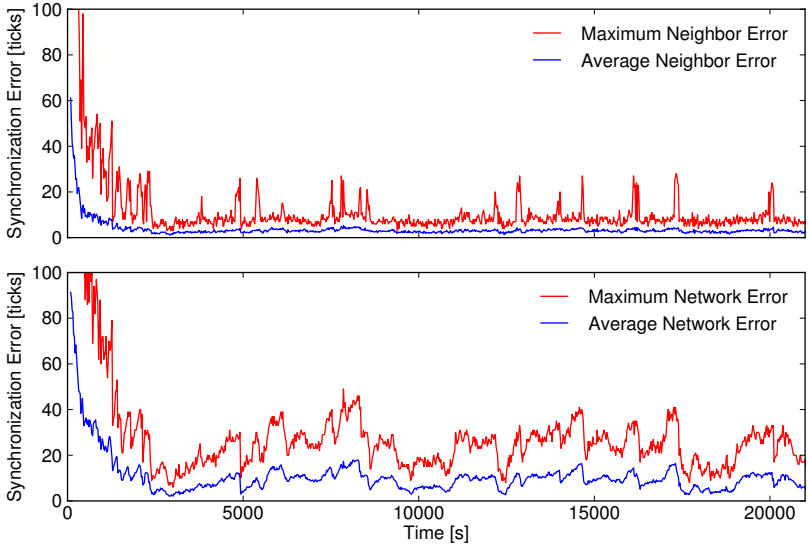


Figure 3.5: Average neighbor ($2.96 \mu\text{s}$) and network synchronization errors ($8.94 \mu\text{s}$) measured for GTSP on a ring of 20 Mica2 nodes (for $t > 3000$ s).

Comparison with FTSP

The same network topology was used to compare the performance of GTSP with FTSP which is considered to be the state-of-the-art time synchronization protocol for wireless sensor networks. The default parameter settings from TinyOS 2.1 were used for FTSP, see Table 3.1.

The measurement results for FTSP on a ring of 20 nodes are shown in Figure 3.6. The time it takes FTSP to synchronize all nodes to the reference node highly depends on the network diameter and the placement of nodes in the network. Again, the time synchronization algorithm is started on all nodes in a random sequence during the first 30 seconds of the experiment.

Protocol parameter	Value
TIMESYNC_RATE [sec]	30
ROOT_TIMEOUT	5
IGNORE_ROOT_MSG	4
ENTRY_SEND_LIMIT	3

Table 3.1: Protocol parameters for FTSP.

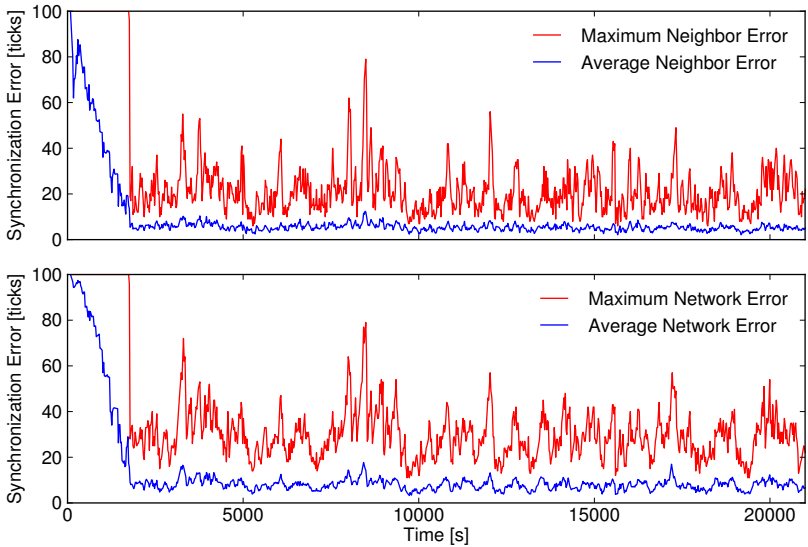


Figure 3.6: Average neighbor ($5.42 \mu\text{s}$) and network synchronization errors ($7.98 \mu\text{s}$) measured for FTSP on a ring of 20 Mica2 nodes (for $t > 3000$ s).

Newly initialized nodes do not send synchronization beacons during an initial period which is determined by the `ROOT_TIMEOUT` parameter. If no other beacons are received during that period, a node declares itself as the new root node and starts broadcasting beacons. Therefore, multiple root nodes are present right after the beginning of the experiment. When a node learns about another root node with a lower identifier than the current root, it switches its root node and adapts its regression table to the logical time of the new root node. If the regression table contains more than `ENTRY_SEND_LIMIT` entries, the node retransmits the logical clock of its current root node. Due to this behavior of FTSP, it takes roughly 30 minutes until all nodes are synchronized to a common logical clock in our setup. We argue that GTSP provides better synchronized clocks during the initialization phase of the algorithm compared to FTSP since clock values are propagated immediately through the network. Although not in the focus of this chapter, this may be an advantage of GTSP in dynamic networks. After FTSP has converged ($t > 3000$ s), we measured an average neighbor synchronization error of $5.42 \mu\text{s}$ and a network error of $7.98 \mu\text{s}$.

FTSP implicitly creates an ad-hoc tree on the network graph by flooding the network with the logical time of the root. Only synchronization beacons containing a higher sequence number are added to the regression table, other

packets are ignored. Therefore, the ring network depicted in Figure 3.7 is split into two subtrees rooted at Node 1. The leaves of these subtrees are Node 8 and Node 15, respectively. Although Node 8 is receiving synchronization beacons from Node 15, this time information is ignored since it contains the same sequence number as previously received from Node 20. Therefore, Node 8 and Node 15 do not synchronize to each other in contrast to the local synchronization approach presented in GTSP.

Figure 3.8 shows the synchronization error between Node 8 and Node 15 for both protocols. Our measurement results show that GTSP provides a better neighbor synchronization compared to FTSP. One might argue that this ring example looks “cooked-up” and that a ring topology does not happen often in practice. While this is true, we insist that the point we make is valid in general, as many reasonable network topologies (e.g., uniform random distribution, grid topology) do not allow a tree embedding with low stretch. In any sensible network topology, FTSP will have neighboring nodes that have a tree distance in the order of the diameter of the network (see Section 3.2). Therefore, the effects shown in Figure 3.8 will always occur in real-world network topologies even though at a smaller scale. Experiments on a 4x5 grid topology with FTSP showed that neighboring nodes can have a large stretch (e.g., we experienced a stretch up to 13) when the node identifiers are assigned randomly and nodes were started in a random order.

3.6.2 Large-Scale Testbed Experiments

While the experimental evaluation on the 20-node Mica2 testbed shows that GTSP provides improved local clock synchronization, this particular network setup has a high diameter but low node density. Furthermore, nodes are placed in close proximity which results in an almost static network topology with high packet reception rates. Therefore, we run experiments on three sensor network testbeds to study the behavior of GTSP on different network topologies and under varying channel conditions.

Setup and Metrics

In the following, we present results from experiments on the DSN [32], MoteLab [151], and TWIST [53] testbeds. For the experiments, GTSP is ported to the TelosB platform [112], which features a MSP430 microcontroller with 48 KBytes of ROM and 10 KBytes of RAM. Unfortunately, the TelosB platform lacks a high frequency quartz crystal such as the 7.37 MHz quartz on the Mica2 platform. Therefore, we use the 32,768 Hz quartz crystal oscillator as the clock source for the synchronization protocol and packet timestamping.

The radio chip used on this platform is the CC2420 transceiver from Texas Instruments, which operates in the unlicensed 2.4 GHz ISM band. To mitigate interference with IEEE 802.11 devices, we used channel 26 during

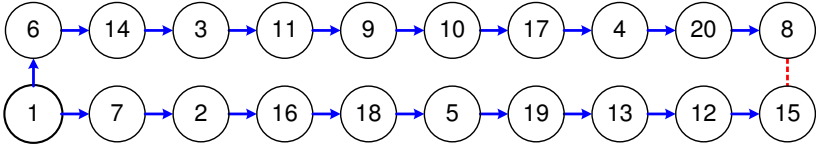


Figure 3.7: The ring synchronization problem: Although Node 8 and Node 15 are direct neighbors in the ring, they are leaves of two different subtrees rooted at Node 1.

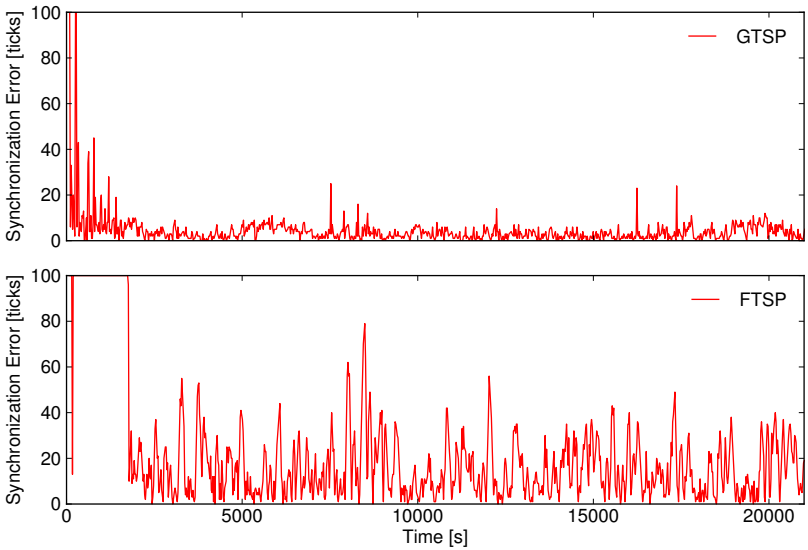


Figure 3.8: Neighbor synchronization error (local skew) between Node 8 and Node 15 on the ring of Figure 3.7 for GTSP (top) and FTSP (bottom). GTSP achieves an average error of $3.34 \mu\text{s}$ with a standard deviation of $2.86 \mu\text{s}$ for $t > 3000 \text{ s}$. FTSP achieves an average error of $15.37 \mu\text{s}$ with a standard deviation of $12.58 \mu\text{s}$ for $t > 3000 \text{ s}$.

Testbed Protocol	DSN		MoteLab		TWIST	
	FTSP	GTSP	FTSP	GTSP	FTSP	GTSP
Neighbor Synchronization Error [ticks]						
Average	1.01	3.55	0.86	1.25	0.92	2.99
Std. dev.	0.88	3.69	0.75	1.12	0.86	2.82
Minimum	0	0	0	0	0	0
Maximum	7	29	4	7	9	36
2-Hop Synchronization Error [ticks]						
Average	1.13	4.03	0.87	2.07	0.97	3.84
Std. dev.	0.97	3.99	0.78	1.98	0.89	3.13
Minimum	0	0	0	0	0	0
Maximum	8	30	4	14	9	40

Table 3.2: Summary of measurement results for FTSP and GTSP on the DSN, MoteLab and TWIST testbeds. A single clock tick on the TelosB platform corresponds to 30.5 μ s.

all experiments. The default transmit power setting of 0 dBm yields an average network diameter of 3 hops in the DSN testbed (34 nodes, avg. node degree 9.45), 5 hops in MoteLab (72 nodes, avg. node degree 10.15), and 2 hops in TWIST (85 nodes, avg. node degree 49.04).

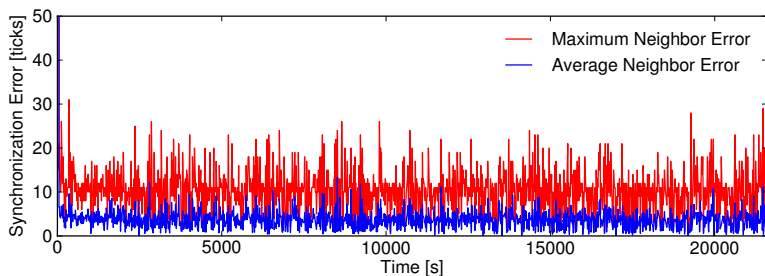
Experimental Results for GTSP and FTSP

Due to the spatial extent of the three testbeds, it is not feasible to employ a single reference broadcast as a network-wide timestamp. Instead, we decided to let all nodes broadcast probe packets following a round-robin schedule. The packet reception is timestamped at the neighbors, and the corresponding logical clock value is collected using the serial backchannel. Therefore, we are only able to measure the current logical clock value within the 1-hop neighborhood of the broadcasting node, which corresponds to the synchronization error between nodes that have a maximum distance of two hops (denoted as 2-Hop Synchronization Error in Table 3.2). For the analysis of the neighbor error in the GTSP experiments we determine a node’s neighborhood based on the entries in its neighborhood table. In the case of FTSP we only include links that have a packet reception rate above 70% in the calculation of the neighbor error.

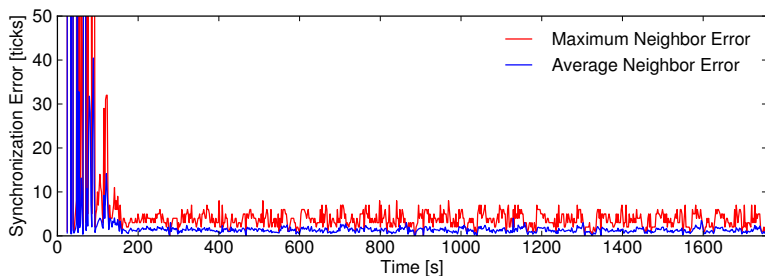
While the duration of an experiment on MoteLab is subject to a quota of 30 minutes, we report measurement results for a 6 hour run on DSN and TWIST (see Figure 3.9). The measurement results for GTSP and FTSP after an initialization time of 1000 seconds are summarized in Table 3.2. The experiments show that the difference in the synchronization accuracy

between FTSP and GTSP is smaller on MoteLab, which has the largest network diameter of all three testbeds. We observe that FTSP provides a similar synchronization accuracy on all testbeds, while the performance of GTSP differs between the three testbeds. We believe that these differences are caused by the characteristic network topology of the three testbeds, which differs along several dimensions, e.g., by network diameter, node density and link stability. In a hierarchical protocol such as FTSP, nodes synchronize their logical clock to a parent node only, while a fully distributed protocol like GTSP incorporates clock estimates from several neighbors, as shown by the graphs in Figure 3.10. In the ideal case, the neighborhood of a node is assumed to be stable and links are reliable, and thus the estimation of the logical clock drift and offset is always based on all neighbors. However, we observe that the neighborhood of a node is changing over time due to unstable links in real-world networks.

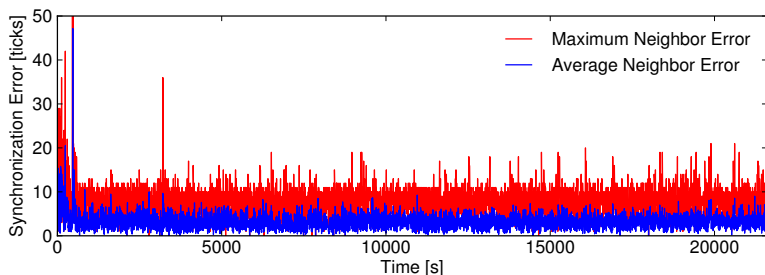
Furthermore, the implementation of GTSP in TinyOS is restricted to a neighbor table with a maximum size of 16 entries. In very dense networks, such as the TWIST network having an average node degree of 49, the averaging operation of relative drift and offset (see Section 3.3) is only based on a small subset of a node's neighborhood instead of all its neighbors due to the limited capacity of the neighbor table. Therefore, it might happen that the resulting network graph is not fully connected, i.e., there is a node that is not included in the table of any node in its neighborhood. Consequently, such nodes will exhibit a constant clock drift to the rest of the network. However, the bootstrap mechanism included in the algorithm for the offset compensation (see Section 3.3.2) will adjust the clock offset accordingly as soon as the clock offset to a neighbor exceeds a certain threshold. Balancing the degree of nodes in the time dissemination graph, e.g., by employing a randomized selection algorithm to insert entries into the neighbor table, is likely to reduce this behavior, however, it is not included in the current implementation of GTSP.



(a) DSN (ETH Zurich), TelosB platform, 34 nodes

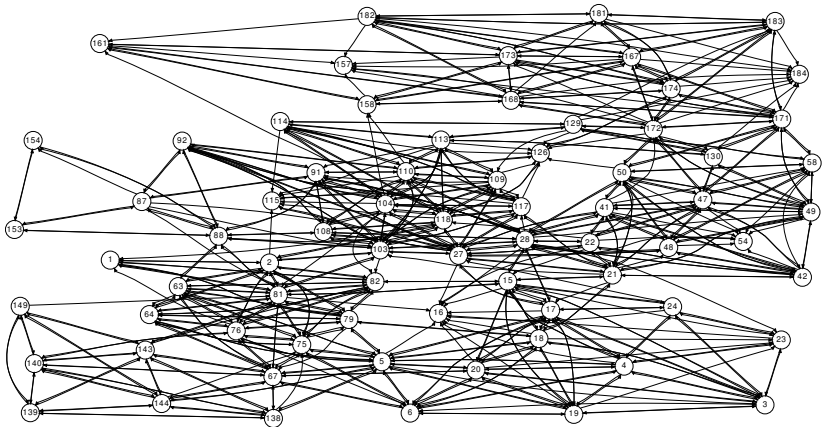


(b) MoteLab (Harvard University), TelosB platform, 72 nodes

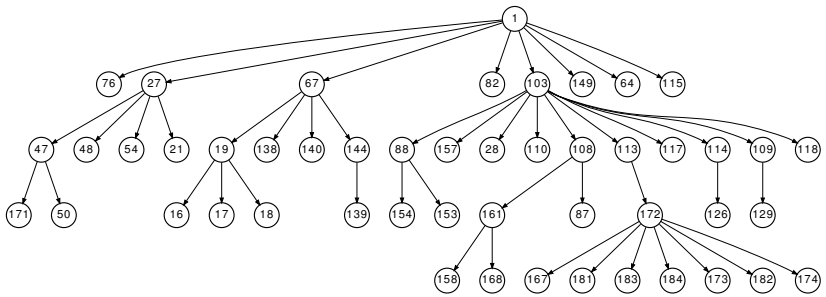


(c) TWIST (TU Berlin), TelosB platform, 84 nodes

Figure 3.9: Summary of measurement results for GTSP on DSN, MoteLab and TWIST. A single clock tick on the TelosB platform corresponds to $30.5 \mu\text{s}$.



(a) GTSP



(b) FTSP

Figure 3.10: Snapshots of the time dissemination graph of GTSP (top) and FTSP (bottom) running on the MoteLab testbed. An edge from A to B indicates that Node B stores an estimation of the logical time disseminated by Node A.

4

Global Clock Synchronization

In the preceding chapter, we assumed that all nodes solely rely on their local hardware clock and do not have access to a reference clock with high accuracy. However, many sensor network architectures consist of a base station that has access to a stable and precise clock reference, for example a GPS receiver. In the absence of an external clock source, the hardware clock of a selected sensor node, the so called reference or root node, serves as the reference. Most clock synchronization protocols are designed to synchronize the clocks of sensor nodes to the reference clock. In this chapter, we will show why existing clock synchronization protocols are not well suited for networks with a large diameter. Based on our observations, we will present the design and implementation of *PulseSync*, a novel clock synchronization protocol. The focus of this work is on the implementation and evaluation of PulseSync, while we refer to [80] and [78] for a detailed analysis of its theoretical background.

4.1 Multi-Hop Clock Synchronization

Wireless sensor networks deployments cover an area of interest, whose spatial extent differs for every usage scenario. While a small network may be within the communication range of a single base station, most scenarios rely on intermediate nodes to forward data to nodes further away. For many applications that collect sensor measurements it is important to have precisely

synchronized clocks even for far-away nodes in the network. For example, the deployment at the Golden Gate Bridge in San Francisco [65] collected sensor readings from 64 nodes attached to the bridge in order to analyze structural vibrations. The resulting network topology had a diameter of 46 hops. Looking at another example, the volcano monitoring network deployed at Reventador in Ecuador consists of 16 nodes placed in a line-like topology around a gateway in the center, which results in a diameter of 9 hops.

Moreover, wireless sensor nodes may be deployed in environments with rapidly changing ambient temperature. The PermaSense project [9] specifies a temperature range from -40 °C to $+65$ °C with changes of up to 5 °C within one minute possible. In such scenarios, a fast response to variations in clock drifts becomes a most pressing issue in the design of a suitable synchronization protocol (see Section 2.3.1).

Consequently, clock synchronization algorithms need to be carefully designed and optimized for operation in multi-hop networks when network-wide synchronization is required. Most algorithms proposed in the literature are primarily designed to synchronize the clocks between two nodes accurately, and this scheme is extended later for multi-hop networks. One node thereby acts as a reference and the other node adjusts its own logical clock based on its estimation of clock drift and offset to the reference clock. This estimation is based on a collection of reference points, of which each contains the global and local timestamps of a synchronization beacon. Once a node has collected a sufficient number of reference points, it becomes synchronized and starts to broadcast synchronization messages on its own, which are then used to synchronize other nodes further away from the reference node.

While such a synchronization scheme can be implemented on mote-class devices, as for example done by FTSP [91], it has also several drawbacks which become visible in larger networks. Clearly, the synchronization accuracy heavily depends on the quality of the reference points collected by a node. Large variations in the message delay or even bit errors in the timestamp contained in the message are incorporated into the node's current estimation of the global clock, and therefore also affect all other nodes that rely on this value. Understanding these effects is of utmost importance when studying algorithms to achieve network-wide clock synchronization. Therefore, we first analyze existing schemes for clocks synchronization using the theoretical framework presented in Section 3.1. The results will give rise to *PulseSync*, a new synchronization algorithm proposed in Section 4.4.

4.2 Linear Regression

Based on the observation that clock drift is fairly constant over a short time period, e.g., within a few minutes, we can assume a linear relationship between the clocks of two nodes. Linear regression or “line fitting” is a well

known statistical method that has relatively low storage and computation overhead, which makes it feasible to implement on current node platforms.

4.2.1 Single-Hop Case

We start with a simplified example, where we have a two-node, single-hop network consisting of node w and v . Assuming a perfect communication channel, no synchronization messages between the two nodes are lost. Furthermore, we assume that both nodes v and w have a clock that is perfectly stable, i.e., the hardware clock rates $h_v(t)$ and $h_w(t)$ are constant but may be different. Under such ideal conditions, it is sufficient for node v to receive two synchronization messages in order to determine its relative logical clock rate l_v and the clock offset θ_v as long as the message delay is known and constant.

However, the time interval between the message is timestamped at the sender and at the receiver is usually not constant but affected by a jitter in the message delay (see Section 2.4). Consequently, it will not be sufficient to estimate the drift and offset based on only two reference points.

Under the assumption that the clock drift remains constant over a short interval of interest, and that the non-deterministic fraction of the message delay is a random variable ϵ , we have the following linear relationship between the hardware timestamp x_i and the logical timestamp y_i forming a reference point p_i :

$$y_i = \alpha x_i + \beta + \epsilon \quad (4.1)$$

The slope α of the line corresponds to the relative logical clock l_v and the axis intercept β is equal to the logical clock offset θ_v . In the following analysis, we assume that the jitter in the message delay is modeled as a random variable ϵ from a normal distribution with zero mean and variance σ^2 .

Linear regression is employed to estimate the drift and offset based on inaccurate timestamps, as illustrated by the fitted line in Figure 4.1.

The estimation of the logical clock is based on the method of least squares [51] according to the linear equation

$$\hat{y} = a + bx.$$

The slope a and intercept b that minimize the sum of the error squares can be calculated as

$$a = \bar{y} - b\bar{x}$$

and

$$b = \frac{\sum_{i=1}^k (x_i - \bar{x})y_i}{\sum_{i=1}^k (x_i - \bar{x})^2}.$$

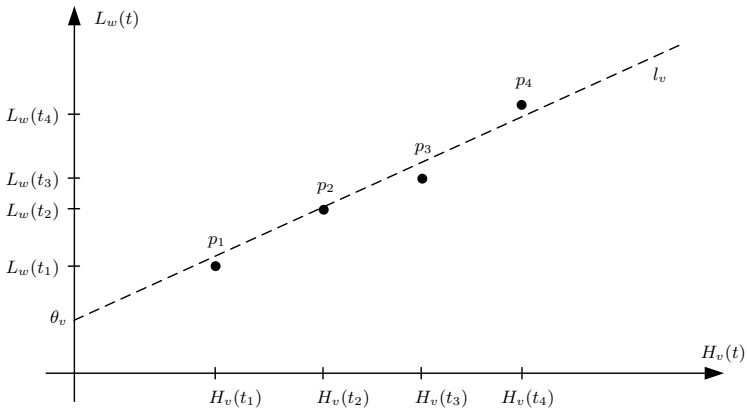


Figure 4.1: Estimation of the relative logical clock rate l_v and offset θ_v between two nodes v and w using linear regression. Timestamps are affected by jitter in the message delay.

Furthermore, it follows that the variance of the fitted value \hat{y}_0 at $x = x_0$ can be calculated as [51]:

$$\text{Var}(\hat{y}_0) = \left(\frac{1}{k} + \frac{(x_0 - \bar{x})^2}{\sum_{i=1}^k x_i^2 - k\bar{x}^2} \right) \sigma^2 = \kappa \sigma^2 \quad (4.2)$$

Consequently, the quality of our estimation for the logical clock value at any hardware clock value x_0 depends on the value of κ shown in Equation 4.2. The parameter κ determines the fraction of the jitter that is passed on to the next hop. We can observe that the variance decreases if the size k of the linear regression table is increased, as depicted in Figure 4.2. Moreover, κ is also independent of the beacon interval B between subsequent timestamps [127]. Therefore, it is only possible to reduce the variance in the estimation of the reference clock if we increase the number of reference points stored in the regression table. The linear regression thereby damps the influence of the jitter in the message delay on the estimation of the reference clock if κ is large. However, both the storage capacity for the regression table and the computation complexity to obtain the linear fit will grow linearly. Moreover, it also becomes more likely that our assumption about constant clock drift during the measurement window of k timestamps will no longer hold due to short-term temperature variations.

As a further observation from Equation 4.2, we note that the variance of \hat{y}_0 increases the farther the current timestamp x_0 is away from \bar{x} , which

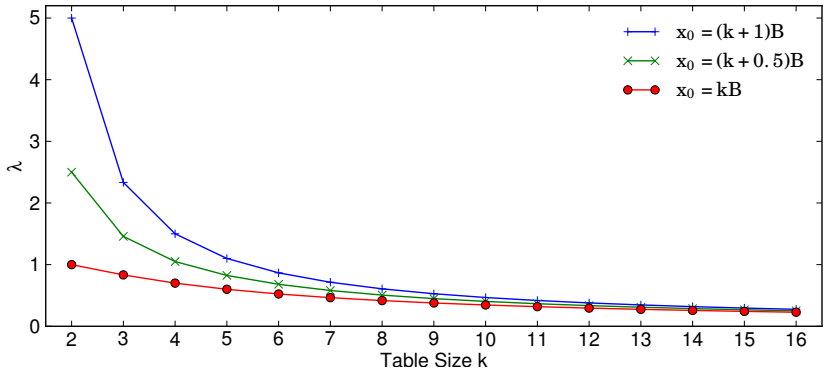


Figure 4.2: Plot of κ , which is the fraction of the variance that is passed on to the next node, for different sizes k of the regression table. In the case of $x_0 = kB$, the estimation of the logical clock is taken immediately after the k -th beacon has been received. Waiting for half ($x_0 = (k + 0.5)B$) or even a full beacon interval ($x_0 = (k + 1)B$) before forwarding the estimation increases its variance.

implicates that we can predict the logical time \hat{y}_0 of a point x_0 in the future or in the past less precisely. As a consequence of this, nodes should not wait for too long before broadcasting a synchronization message to other neighbors. This effect leads to an amplification of the jitter in the message delay for small sizes of the regression table, as shown in Figure 4.2.

4.2.2 Multi-hop Case

Having obtained a sufficient number of reference points for the logical clock, a node becomes synchronized and can serve itself as a reference clock for its neighbors. We look at a simple example network of 3 nodes (v , w and u), where v is the reference node and time is propagated from v to w and then from w to u . For this purpose, an estimation \hat{y}_w of the current logical time is forwarded by node w based on the slope and intercept from its regression table. As discussed before, the variance of the timestamp \hat{y}_w grows according to Equation 4.2 as farther as the hardware clock value x departs from \bar{x} .

Furthermore, the timestamp y'_u measured at node u is also affected by a random jitter σ in the message delay, which is statistically independent of the variance in the estimation \hat{y}_w by node w .

However, the problem with this approach is that the synchronization accuracy depends on the dynamic behavior of the system. Due to the limited storage capacity of the regression table, only the last k samples are taken

into account to estimate clock drift and offset. This has the effect that the current slope may change whenever a new beacon is received. Moreover, a single beacon which is severely affected by jitter introduces a large error in the timestamp forwarded with the next k successive beacons.

4.3 The Flooding Time Synchronization Protocol

In the following, we will take a closer look at the properties of the Flooding Time Synchronization Protocol (FTSP) [91], which is being widely adopted by the WSN community for time synchronization. After the initialization phase, FTSP utilizes a spanning tree rooted at the root node r to disseminate information on r 's state throughout the network. Every node tries to synchronize its logical clock as good as possible to the one of r , while adjusting its clock value whenever an offset is detected. To this end, nodes broadcast their estimate of the reference clock value once every beacon period B . They store the k most recent values received from their parent in a table and determine their current estimate of r 's clock value and the relative clock rate by means of a linear regression of this data set. Furthermore, when receiving a new message from their parent, they immediately update their logical clock rate and offset.

4.3.1 Analysis of FTSP

A critical weakness of the protocol is the fact that errors introduced by jitter affect not only clock *values*, but also all clock *rates* further down the tree. Since the nodes send messages without following a predefined schedule, the expected propagation speed of time information is merely one hop in $B/2$ time. Meanwhile, clock skew can accumulate, as illustrated in Figure 4.3 for the special case $k = 2$. Nevertheless, our simulation results show that the same problem occurs in case of reasonable sizes of the regression table (see Figure 4.4). FTSP is simulated on a line topology, the root was fixed to be the first node of the line. The jitter in the message delay is a normally distributed random variable with zero mean and variance of 1 clock tick. Furthermore, there is no hardware clock drift, i.e., the errors introduced in the system are only due to the message delay jitter and the linear regression. The original implementation of FTSP in TinyOS ignores estimates that are seemingly invalid due to large skews; it even resets the regression table if this happens repeatedly (cf. Section 4.6). This has been suppressed in the simulations to demonstrate the behavior of the basic protocol for large diameters.

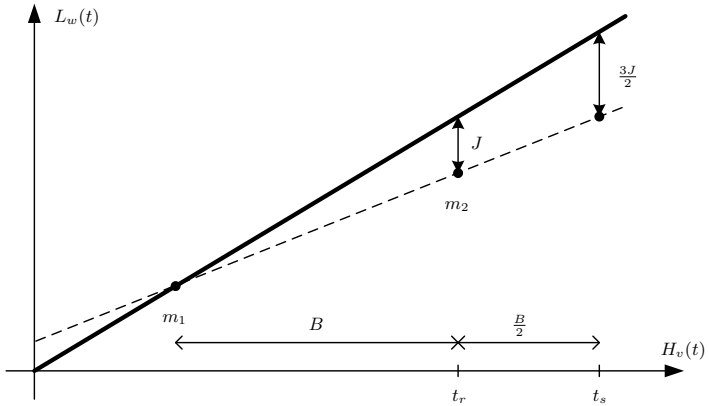


Figure 4.3: Logical clock computation scheme by linear regression for the special case $k = 2$. Errors introduced by jitter are amplified because they affect both the axis intercept and the slope of the regression line.

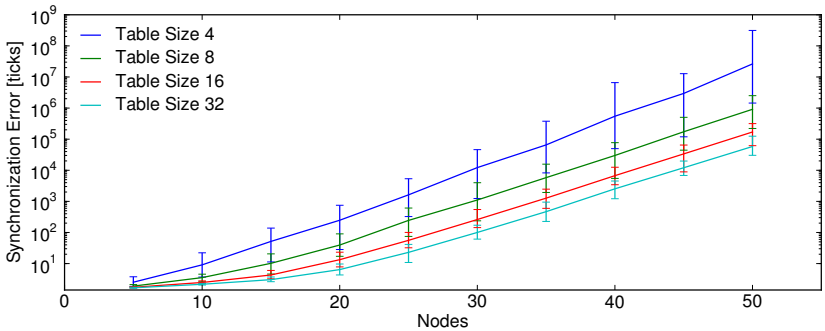


Figure 4.4: Simulation of FTSP for line topologies with different diameters using varying sizes $k = [4, 8, 16, 32]$ of the regression table. Mean synchronization errors averaged over 50 runs, error bars indicate values of runs with maximum and minimum outcome.

4.4 The PulseSync Protocol

In the previous section, we have observed two major issues that may degrade the quality of (global) synchronization:

- When forwarding logical clock values the estimation of the clock rate should not depend on the most recently received clock value. Otherwise we can observe a self-amplification of errors.
- If sending times are not aligned, information propagates slowly, giving time to pile up skew originating from small differences in clock rates.

Both of these problems are already present if the system is static. Certainly, to cope with dynamics such as changing hardware clock rates or network topologies, quick dissemination of information is also a highly desirable property of an algorithm. Hence, in this section we will give a high-level description of an algorithm not suffering from these drawbacks. In Section 4.6 the details of the implementation in TinyOS will be discussed.

Forwarding Scheme

The basic idea of the algorithm is to distribute information on clock values of a reference clock as fast as possible, while minimizing the number of messages required to do so. In particular, we want nodes to send messages only once in B time, while avoiding that it takes in expectation $\frac{B \cdot D}{2}$ time until distant nodes affect each other. Thus, we propose to flood a “pulse” through the network, implicitly building a breadth-first search tree, as shown in Figure 4.5. The root node r of the tree becomes the reference node, which can be selected using any leader election algorithm. In our case, we assume that nodes have unique identifiers, and that the node with the smallest identifier is the root node, similar as in FTSP.

The inherent problem with this scheme is that it obviously provokes collisions when nodes try to forward a message concurrently. Therefore, an implementation of PulseSync in a wireless network will have to schedule the flooding and nodes may have to wait before they can send a message. We discuss this issue in more detail in Sections 4.6.5 and 4.7.1.

Drift Estimation

Although PulseSync strives to forward the estimation of the root’s clock value \hat{L}_r as quickly as possible, we need to compensate for the local processing delay between reception (t_{rx}) and forwarding (t_{tx}) of a message. As shown before, increasing received estimates at the speed of the own logical clock, which also incorporates this estimate, may result in self-amplification of errors as in FTSP, yet to a smaller extent, as nodes forward estimates as

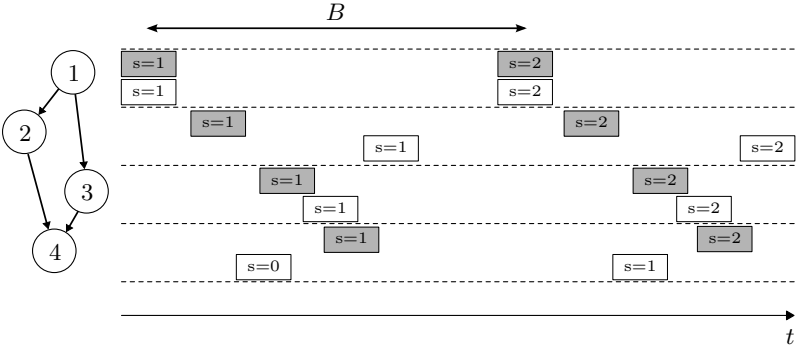


Figure 4.5: Timeline of message transmissions for PulseSync (filled boxes) and FTSP (empty boxes) on a 2-hop network of four nodes. In FTSP, each node maintains its own schedule and a synchronization message is sent every B seconds. In contrast, PulseSync employs coordinated flooding of synchronization messages through the network. Upon reception of a new message with a higher sequence number s , a node sends its own synchronization message as soon as possible.

quickly as possible. Similar to our approach, the Rapid Time Synchronization Protocol (RATS) employs fast flooding of synchronization beacons using the Elapsed Time of Arrival (ETA) primitive [72, 71]. However, no drift correction is applied to the logical timestamp that is inserted in the forwarded synchronization beacon. Relying on the bare hardware clock rates for the update of the timestamp may result in errors comparable to (or even dominating) the jitter. However, as the time between reception and forwarding of an estimate will be small compared to the beacon interval B , virtually any estimate \hat{r}_v^r a node v acquires of the relative clock rate to r will do, as long as we avoid the self-amplification issue. Therefore, nodes compute an approximation of r_v^r by means of the first k messages they receive, while relying on their hardware clock (i.e., estimating relative rate 1) when forwarding the first k messages. A similar approach has been used for high latency networks where the drift during message transfer is a major error source [144]. Thus, we neither slow down the initialization process nor do we sacrifice accuracy if the topology is stable. The logical clock value included in the forwarded message is calculated as

$$\hat{L}_r(t_{rx}) = \hat{L}_r(t_{rx}) + (H(t_{tx}) - H(t_{rx})) \cdot \hat{r}_v^r.$$

While nodes forward the reference clock value as quickly as possible, they also employ a drift compensation mechanism to provide an accurate logical

clock between two synchronization beacons. To even out the effects of the random jitter, this estimate is based on k values. This can be done in several ways, e.g., by a moving average filter [137], or by linear regression. Note that the logical clock values calculated using linear regression are only used locally and are not forwarded to other nodes as in FTSP.

4.4.1 Analysis of PulseSync

A detailed theoretical analysis of the PulseSync algorithm is presented in [80] and [78]. In summary, we obtain a concentration result stating that the expected clock skew of PulseSync in distance d from the root node is bounded by $\mathcal{O}(\mathcal{J}\sqrt{\frac{d}{k}})$ with high probability. Consequently, the synchronization error grows when the hop count d from the reference node increases. Thus, we expect that the network-wide synchronization error grows with the square-root of the diameter. On the other hand, the synchronization accuracy is improved when the estimation of the reference clock is based on larger table sizes k .

4.5 Simulations

We implemented our PulseSync algorithm in the Sinalgo network simulator [44] to study its properties on networks with large diameters, which exceed the dimensions of current WSN testbeds. Hardware clock drift is uniformly distributed between -40 and $+40$ ppm for each node, but stays constant during a simulation run. The jitter in the message delay is drawn from a normal distribution with zero mean and variance $1.0 \mu\text{s}$. Synchronization pulses are immediately forwarded by intermediate nodes. Extensive simulations on line topologies with a diameter between 10 and 1000 hops show that both the local and global skew grows less than linearly, see Figure 4.6, as predicted by the theoretical analysis [80].

4.6 Experimental Evaluation

We present experimental results for PulseSync to verify that our theoretical results also apply in practice. To that end, we have implemented PulseSync on the Mica2 platform. Furthermore, we compare our approach to FTSP, the state-of-the art clock synchronization protocol for wireless sensor networks.

4.6.1 Hardware Platform

The hardware platform used for the implementation of PulseSync is the Mica2 sensor node [57]. The Mica2 platform features an Atmel ATmega128L microcontroller with 4 KBytes of RAM, 128 KBytes of program ROM and

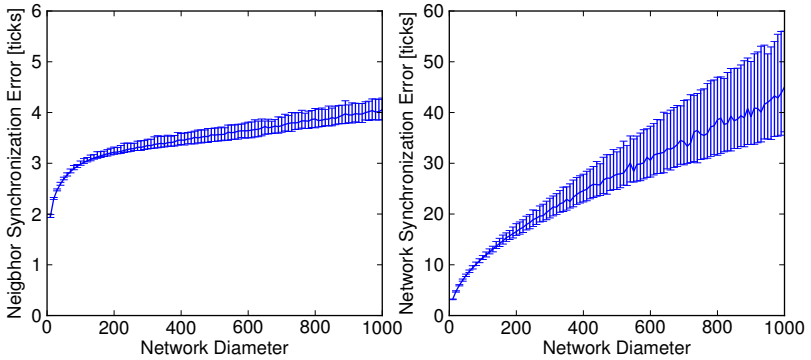


Figure 4.6: Simulations of the PulseSync algorithm on line topologies with different diameters. Jitter is normally distributed with variance $1 \mu\text{s}$, clock drift up to ± 40 ppm, time between pulses is $B = 30$ s. Mean synchronization errors are averaged over 50 runs, error bars indicate values of runs with maximum and minimum outcome.

512 KBytes of external flash storage. The Texas Instruments CC1000 radio module offers data rates up to 76.8 kBaud using frequency shift keying (FSK). We use the 7.37 MHz quartz oscillator as the system clock source, and configure a 16-bit timer to operate at $\frac{1}{8}$ of the oscillator frequency, resulting in a clock frequency of 921 kHz. An additional 16-bit variable keeps track of counter overflows, providing us in the end with a 32-bit hardware clock which offers a precision of roughly a microsecond.

4.6.2 PulseSync Implementation in TinyOS

The implementation of the PulseSync protocol on the Mica2 platform is done using TinyOS 2.1. The protocol implementation provides access to a synchronized global clock for other components of the application running on the node.

After startup, the *PulseSync* component is initialized by setting the logical clock to the current hardware clock value. By overhearing the channel for other synchronization messages, a node will quickly learn about the presence of a reference node. If no synchronization messages have been received for a certain time, the node will declare itself as the reference node and starts to broadcast periodic synchronization pulses. A node will receive periodic pulses from nodes located closer to the reference node. If the reference node should suffer from a hardware failure or the node gets disconnected, no new sync pulses will be received. When no synchronization messages are received by a node for several consecutive beacon intervals, the node starts again to

advertise itself as the reference node.

Depending on the network topology, a node will receive synchronization messages from multiple nodes in its neighborhood. Only the first pulse arriving at a node will be forwarded. It is very likely that this pulse has traveled on a shortest path from the root node and, therefore, the jitter accumulated along this path is assumed to be smaller than on longer paths. To detect duplicate pulses, we insert a sequence number field into the synchronization message which is increased by the reference root after each sent pulse.

Synchronous Acknowledgments

Clearly, the performance of any clock synchronization protocol is degraded when synchronization messages do not reach their destination due to a lossy communication channel. This is a problem each protocol has to cope with in a real-world deployment. However, in our testbed experiments we are interested in the actual performance of our clock synchronization protocols, PulseSync and FTSP. To mitigate the effects of packet loss, we implemented synchronous acknowledgments on the application layer for both protocol implementations.

4.6.3 Testbed Setup

We use an indoor testbed of 20 Mica2 sensor nodes which are placed in close proximity, thus forming a single broadcast domain. In order to explore the behavior of FTSP and PulseSync in a network with a large diameter, we enforce a line topology in software. A base station node connected to a PC is used to monitor the experiment. To test the synchronization accuracy of the two different clock synchronization protocols, we rely on external events triggered by the arrival time of special probing packets. On reception of such a time probe, each node writes both its hardware and logical time to the external flash memory. At the end of the experiment, the measurement results stored in the flash memory of the nodes are transferred to the PC for further analysis. The time interval between time probe events is uniformly distributed between 18 and 22 seconds. This guarantees that the time interval between the clock synchronization packets and the arrival of the time probes is changing continuously.

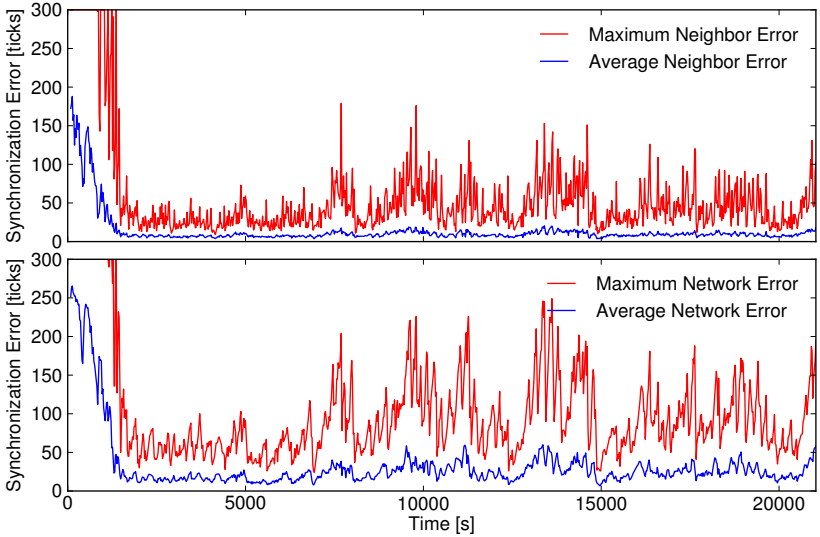
We used the FTSP implementation available from the TinyOS 2.x CVS repository during our experiments. Using the default parameter settings, FTSP failed to synchronize all nodes in the network to the reference node even after a long time period. We observed that it takes many rounds until all nodes agreed on the same reference node. After all nodes have been started, it takes `ROOT_TIMEOUT` rounds until multiple nodes in the network start to claim being the reference node by broadcasting synchronization beacons. Then, these new reference nodes will ignore beacons from other nodes

having a lower identifier for the next `IGNORE_ROOT_MSG` rounds. If a node learns about a new reference node with a lower identifier, it will wait until it is synchronized well enough to this new reference node by listening to `ENTRY_SEND_LIMIT` beacons before it will forward beacons itself. Another issue with FTSP is the fact that nodes will clear the content of their regression table when the synchronization error between the received beacon and the estimated time of the node exceeds the `ENTRY_THROWOUT_LIMIT` parameter. Such a node will stop sending its own synchronization beacons until it has re-established synchronization to its reference node again. However, if the local skew incurred by the protocol gets large, this behavior does not protect against infrequent outliers or failures, but contrariwise worsen the situation. The next node in the line will think that the current reference node died and it will start a new leader election round by advertising itself as the new root node. Since such effects will clearly degrade the performance of any clock synchronization protocol, we slightly modified FTSP to work with a fixed network topology (node 1 is the reference node). Furthermore, we set `ENTRY_THROWOUT_LIMIT` to the maximum integer value to prevent nodes from clearing the regression table. For a fair comparison, we also enforced a fixed topology with Node 1 as the root for PulseSync.

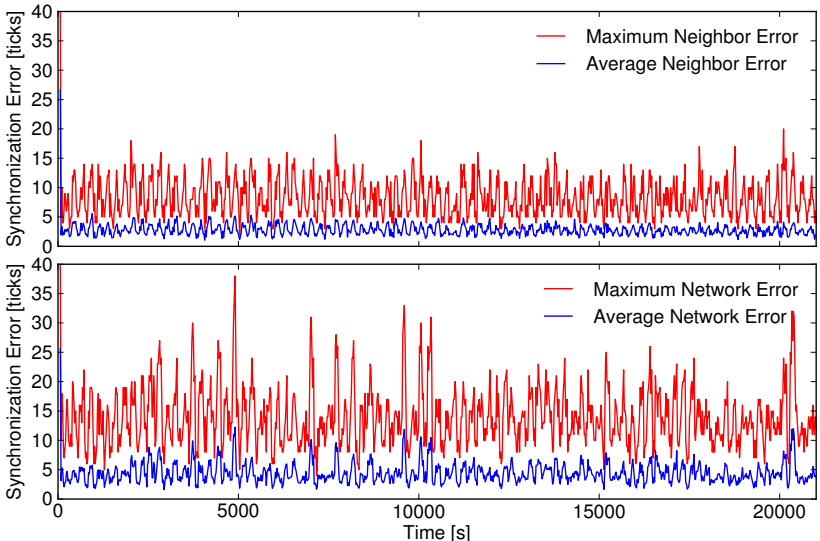
4.6.4 Measurement Results

For both PulseSync and FTSP, we collected measurement results during a 6 hour run. This resulted in roughly 20,000 data points collected from the sensor nodes. In our evaluation we focus on two different metrics: the *local skew* and the *global skew* (as defined in Section 3.1) are calculated for each time probe event. We are particularly interested in the maximum global skew so that we can get an impression on the worst-case accuracy of both clock synchronization protocols.

Figure 4.7(a) shows the measurement results for FTSP on a line of 20 Mica2 nodes. It can be seen that it takes FTSP roughly 2000 seconds until all nodes are synchronized. Measurement results for the implementation of PulseSync are presented in Figure 4.7(b). PulseSync converges rapidly to a common logical time since the time information is flooded in a single round from the reference node to all other nodes throughout the network. Although PulseSync and FTSP have the same message complexity (one message per node and synchronization period), the fast flooding approach of PulseSync achieves a significantly improved synchronization accuracy on the same network topology. Furthermore, our experiments confirm our finding that the synchronization error to the root node increases exponentially when using FTSP, while PulseSync performs significantly better (see Figure 4.8). The measurement results are summarized in Table 4.1. We can see that the synchronization error on the real sensor node hardware is consistent with what we expect from the simulations, although it seems that the simulations



(a) FTSP



(b) PulseSync

Figure 4.7: Neighbor and network errors measured for FTSP (top) and PulseSync (bottom) on a line topology of 20 Mica2 sensor nodes.

are based on too optimistic assumptions about clock drift and/or jitter. In particular, we assumed perfectly stable hardware clocks, but at the level of accuracy provided by PulseSync even small fluctuations in the clock rates become visible (cf. Figure 2.1).

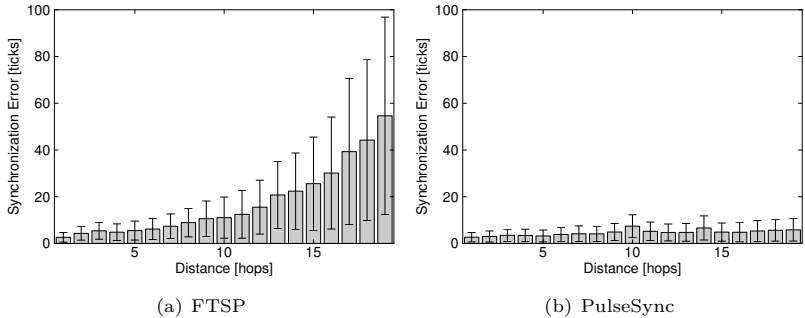


Figure 4.8: Synchronization error versus distance from the root node.

	FTSP	PulseSync
Average Network Error [μ s]	23.96	4.44
Maximum Network Error [μ s]	249.00	38.00
Average Neighbor Error [μ s]	9.04	2.79
Maximum Neighbor Error [μ s]	129.00	20.00
Synchronization messages	13,510	13,504

Table 4.1: Measurement results for a line of 20 Mica2 sensor nodes.

4.6.5 Large-Scale Testbed Experiments

To demonstrate the applicability of our approach on sensor networks with arbitrary topologies, we evaluated PulseSync on different testbeds. We decided to implement our protocol on the TelosB platform [112], which is available on the DSN [32], MoteLab [151] and TWIST [53] testbeds. However, the lack of a high-frequency crystal quartz on the TelosB platform decreases the clock resolution to 30.5 μ s when using the 32 kHz real-time clock. Therefore, the jitter in the message delay of the CC2420 radio, which exhibits a standard deviation of 0.040 μ s (see also Section 2.4), has a much smaller influence on the timestamp of a synchronization beacon compared to the Mica2 platform.

Scheduling of Synchronization Beacons

Due to the high node density observed on all three testbeds, collisions are very likely to occur when multiple nodes try to forward the synchronization pulse roughly at the same time. Therefore, upon receiving a new pulse from an upstream neighbor, a node waits for random backoff time uniformly distributed between 0 and 256 ms before it initiates the packet transmission. In addition to this suspension mechanism, the TinyOS implementation for the CC2420 radio performs a clear-channel assessment (CCA) before the transmission is started. Although this simple backoff mechanism seems to mitigate the occurrence of collisions, it prolongs the time span between receiving and forwarding a packet. Thus, we will briefly discuss several approaches to improve the scheduling of beacons in Section 4.7.

Experimental Results for PulseSync and FTSP

At the beginning of each experiment, the synchronization protocol is started uniformly at random within the first 16 seconds. During the whole experiment, nodes transmit radio packets following a round robin schedule serving as a reference point which is timestamped with the estimated logical clock at the receivers. However, such a probe packet will reach only all 1-hop neighbors of the sender and cannot serve as reference broadcast for the whole network. Therefore, there is no mechanism to obtain the current logical timestamp of all nodes simultaneously.

We compared the performance of PulseSync and FTSP on the three testbeds. The results summarized in Table 4.2 demonstrate that the average synchronization error of PulseSync is only 59–81% of the error observed with FTSP. We argue that the difference between the two protocols would be more distinctive on a testbed with a large diameter, as predicted by simulations and as manifested on the Mica2 testbed. Another benefit of the forwarding mechanism in PulseSync is the shortened initialization phase as visible in the transient behavior depicted in Figures 4.9, 4.10 and 4.11. Assuming ideal conditions with no packet loss and collisions, all nodes have received an estimation of the logical clock value after the first pulse, which is enough to calculate their logical clock offset. Immediately after, the logical clocks start to drift apart due to drifting hardware clocks. After the second pulse, nodes obtain a rough estimation of the logical clock rate, which is gradually improved when subsequent pulses arrive. Therefore, PulseSync achieves synchronization in $\mathcal{O}(D)$ time. In contrast, it will take FTSP significantly longer to propagate time information within the network, which is visible as a large synchronization error during the first few minutes of the experiment. As a prerequisite for a node to become synchronized, it will first have to collect a sufficient number of synchronization beacons. Therefore, it will take $\mathcal{O}(BD)$ time until all nodes are synchronized.

Testbed Protocol	DSN		MoteLab		TWIST	
	FTSP	PulseSync	FTSP	PulseSync	FTSP	PulseSync
Network Synchronization Error [ticks]						
Average	1.13	0.67	0.87	0.71	0.97	0.59
Std. dev.	0.97	0.66	0.78	0.68	0.89	0.61
Minimum	0	0	0	0	0	0
Maximum	8	9	4	4	9	8

Table 4.2: Summary of measurement results for FTSP and PulseSync on DSN, MoteLab and TWIST, after the initialization phase of 1000 seconds.

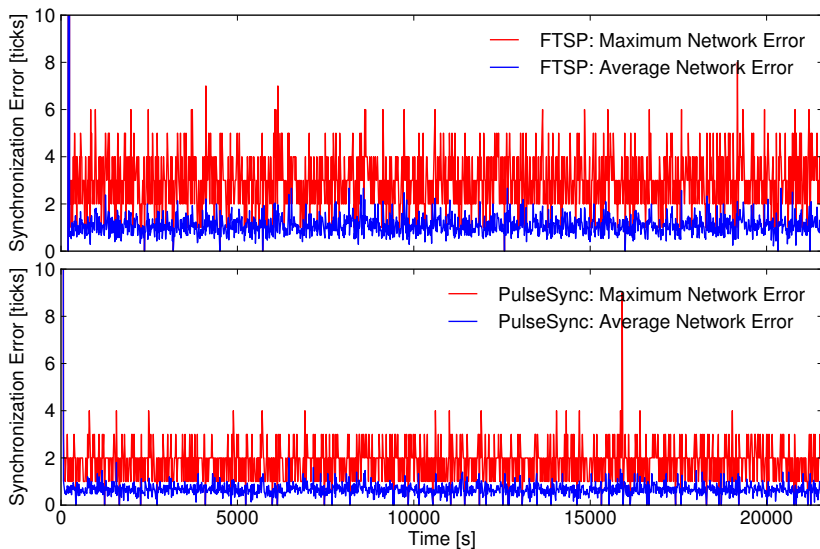


Figure 4.9: Measurement results for the DSN testbed.

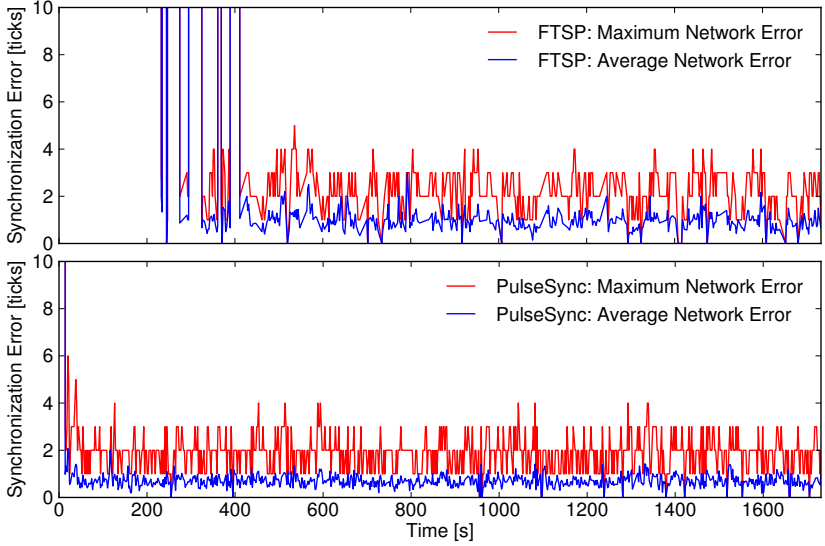


Figure 4.10: Measurement results for the MoteLab testbed.

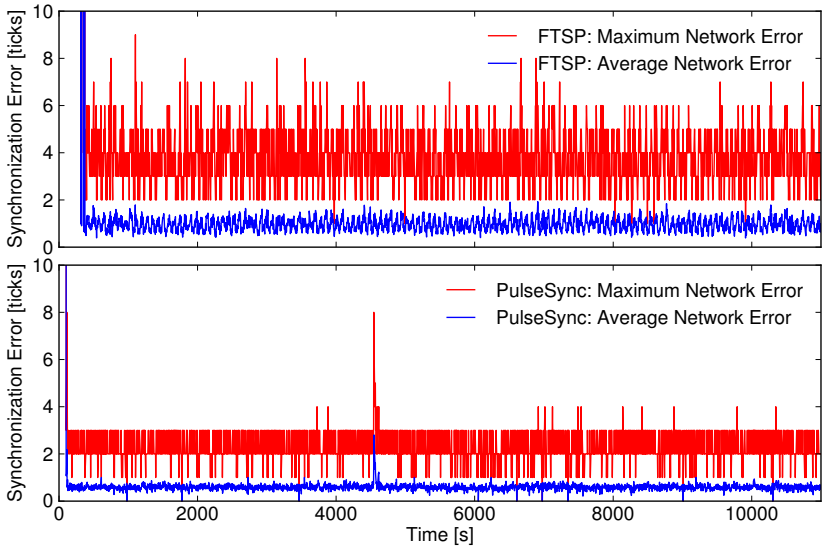


Figure 4.11: Measurement results for the TWIST testbed.

4.7 Protocol Improvements

In this section we discuss three improvements to the basic PulseSync algorithm. First of all, an efficient broadcasting mechanism is vital for more complex and dense topologies. Moreover, we argue that the fast flooding mechanism of PulseSync works well together with duty-cycling techniques to reduce the power consumption of resource constrained nodes. Finally, we discuss some approaches to further reduce the local skew.

4.7.1 Efficient Flooding in Wireless Networks

As we learned in this chapter, time information must be propagated as quickly as possible. In a wired network, PulseSync can be implemented directly by a simple flooding algorithm. In wireless networks, scheduling the transmissions is more difficult because of interference. This issue has been studied arduously in the theory community; it is generally known as the broadcast problem in radio networks.

More than 20 years ago, it was shown that finding an optimal broadcast schedule is NP-hard [23]. One may approximate the optimal solution [22], however, at a communication cost which is clearly beyond anything practically tolerable. In addition, these early centralized broadcast solutions are designed for graphs, and graphs do not model wireless transmissions well [101].

More promising are distributed algorithms on unknown topologies. Bar-Yehuda et al. [6] show that a straightforward randomized protocol will reach all nodes with high probability in $\mathcal{O}(\mathcal{T}D \log^2 n)$ time. This result can even be improved slightly [26], matching a known lower bound.

One may improve the broadcast time even more by making the graph sparse by computing a connected dominating set [130] first. After receiving the PulseSync message, nodes in the connected dominating set simply transmit with constant probability, logarithmically often. Assuming that the interference radius is only a constant factor larger than the transmission radius, this broadcast algorithm finishes in $\mathcal{O}(\mathcal{T}D \log n)$ time with high probability.

In the context of wireless sensor networks, the ability to quickly flood packets through the network is a fundamental primitive for various network protocols and applications, such as data dissemination [82], time synchronization [91, 121, 128, 127] or network reprogramming [59]. Several protocols have been proposed to increase the reliability of flooding, e.g., the *Robust Broadcast Protocol (RBP)* [141] or *Collective Flooding (CF)* [164] which exploit information about the 1-hop neighborhood of a node to achieve this.

Other approaches employ concurrent transmissions by different nodes to reduce the latency of flooding. Thereby, the Flash flooding protocol [89] exploits the power capture effect, where a receiver is able to successfully

decode the strongest packet amongst multiple concurrent transmissions [156]. Recently, Ferrari et al. [42] showed that it is feasible to synchronize the packet transmissions from multiple senders so that they will interfere constructively with high probability. Based on this approach, they propose *Glossy*, which provides a flooding latency of a few milliseconds for networks with a small diameter.

Although we did not employ these sophisticated algorithms in our prototype implementation, they demonstrate that efficient solutions of the problem are feasible. With such techniques, for a reasonable choice of B (30 seconds or more) it should be possible to complete pulses in less than B time on any real-world topology; even if this was not the case, a new pulse could be started before the previous one is finished. In B time the nodes that are active because of the current pulse will be far away from the root, hence this will not cause additional collisions. In this scenario it suffices to complete pulses within kB time to maintain the sub-linear growth of global skew with respect to the diameter.

Because of additional external noise and interference, one shall not rely on broadcasting algorithms alone. In practice, minimizing the broadcast time is only one side of the coin, reaching most nodes is important as well. Indeed, if a bridging node misses out a pulse, all nodes on the wrong side of the bridge will do as well. For this reason one adds a pulling mechanism to the broadcast algorithm. If a node does not receive the time information during a pulse, it will actively ask its neighbors. The reliability of this approach is apparent from the results provided in Section 4.6, as it solved the problem for the examined line topology: Sending a positive acknowledgement on reception of a message corresponds to sending a request—i.e., a negative acknowledgement—when a message is overdue.

4.7.2 Low-Power Operation

Energy is one of the limiting factors one has to keep in mind when designing applications and protocols for sensor networks. We identify three major sources of energy consumption in the context of clock synchronization: hardware clocks, message exchange and local computation.

Message Exchange

Current time synchronization algorithms, e.g., FTSP, broadcast periodic beacons at a certain rate (e.g., every 30 seconds). However, the time point when a node broadcasts the next time synchronization beacon is not coordinated with its neighbors. One advantage of this approach is its simplicity since neighboring nodes do not need to agree on a common schedule when beacons can be sent or received. However, the drawback of uncoordinated sending

Radio	Transmit mode	Receive mode	Idle mode
	[mA]	[dBm]	[μ A]
CC1000	16.5	0	0.2
CC2420	17.4	0	0.02
XE1205	33.0	5	0.2
RF230	14.5	1	0.02

Table 4.3: Current consumption for different radio transceivers.

becomes apparent when it comes to low-power operation scenarios since a node needs to operate its radio in the receive mode, see Table 4.3.

In contrast, PulseSync simplifies low-power operation since pulses will arrive only during a specific and short time interval. This allows to operate the radio chip at a very low duty-cycle. As long as a node is synchronized to the rest of the network, it can easily calculate when the next synchronization pulse will arrive and switch on the radio accordingly. This allows for a temporal decoupling of sensing, computation and communication tasks, as proposed in the slotted programming approach [45].

Hardware Clock Accuracy

Depending on the clock frequency and on the width of the clock register, the timer will overflow frequently. A 16-bit timer sourced by a 32 kHz crystal oscillator will overflow every two seconds, thus having the same hardware clock value every two seconds. Therefore, by using a 16-bit wide counter register it is only possible to distinguish the timestamps of two different events that occur during an interval of two seconds. Keeping track of counter overflows in the upper 16 bits gives us a 32-bit wide hardware clock. This 32-bit hardware clock will wrap around only every one and a half day which should be fairly enough for most applications. Operating a continuously running hardware clock is energy consuming even when operating the microcontroller in a low-power mode for most of the time. Although the clock register and the crystal oscillator can still be operating in certain low-power modes, the counter overflow cause the microcontroller to power up frequently in order to execute the corresponding interrupt handler.

The situation becomes even worse if a hardware clock with a higher granularity is required by the application. The Mica2 hardware platform is equipped with an additional quartz crystal of 7.37 MHz nominal frequency. Using this crystal oscillator with a pre-scaling factor of 8 will result in a hardware clock with roughly 1 μ s (921 kHz) clock granularity but coming at the expense of more than 13 timer overflows per second.

Schmid et al. [129] propose a duty-cycling approach for hardware clocks to eliminate the tradeoff between low-power operation and high clock accuracy.

A low-frequency clock which is running continuously provides a base line for the hardware time. If a higher clock granularity is needed temporarily, e.g., to timestamp the reception of a radio packet, a higher-frequency clock is used to increase the timestamping accuracy between the clock ticks of the low-frequency clock.

4.7.3 Local Clock Skew

So far we restricted our attention to the global skew, leaving the local skew aside. This comes from the fact that the fast propagation of clock estimates attained by the PulseSync algorithm in some sense makes the global skew “local”. While conventional synchronization algorithms rely on techniques where information propagates at merely one hop per beacon interval, PulseSync floods it through the whole network within a single pulse. Thus, global skew cannot be accumulated for $\Omega(DB)$ time before being recognized by distant nodes. Nonetheless, it grows like \sqrt{D} due to the increasing (total) uncertainty in delays due to the jitter, whereas apparently neighbors are able to synchronize both their clocks and clock rates more precisely. The tree-like communication structure of PulseSync does not exploit this knowledge and therefore the full global skew may be observed between neighboring nodes which are at distance D (or even $2D$) in the tree.

Opposed to that, GTSP (see Chapter 3) exploits all local information by construction, as it strives to *locally* optimize skews in each step. However, this comes at the price of slow dissemination of information, therefore incurring a potentially large global skew. In theory, this discrepancy between optimizing locally and globally can be overcome easily. If we do not care about message complexity, but just frequency of pulses, *every* node could act as root of its own flooding tree. Clock values could then be computed as a weighted average over estimates of all nodes, which then would both locally and globally agree very well. However, this approach would require n^2 messages per pulse, which might not be acceptable for energy-constrained applications. Using a centralized approach, the root could collect information on the clock skew on each *edge* in the graph by means of a flooding-echo protocol. This comes at message cost $2n$ since a single message can be received by all neighbors simultaneously. The root then computes optimal skew and rate corrections in a centralized manner, where “optimal” can be any desired quality measure: global skew, local skew, sum of squared errors on each edge, etc. Finally the obtained corrections are distributed in a second flooding phase. Although such an approach would provide improved local and global clock skews, the size of the messages would become prohibitively large when relying on such a scheme.

5

Conclusions

Sensor network applications can greatly benefit from synchronized clocks to perform various tasks such as data fusion or energy-efficient communication. A perfect clock synchronization algorithm should fulfill a handful of different properties at the same time: precise global and local time synchronization, fast convergence, fault-tolerance, and energy-efficiency.

Classical time synchronization algorithms used in wireless sensor networks strive to optimize the *global* clock skew. All nodes within the network should be synchronized as accurately as possible to a common reference clock. However, due to the fact that clocks exhibit drift and time transfer using radio packets includes jitter in the message delay, the synchronization error of every global clock synchronization algorithm is expected to increase with increasing hop count from the reference node. Consequently, neighboring nodes can be affected by a large mutual clock synchronization error if the clock synchronization beacons they receive from the reference node are passed along different paths from the reference node. Thus, we argue that many practical sensor network applications will benefit from minimizing *local* clock skew. Application scenarios that greatly benefit from low local clock error include for example medium access protocols or sensor fusion. Therefore, we presented the *Gradient Time Synchronization Protocol (GTSP)* in Chapter 3, which is a fully distributed time synchronization protocol designed to minimize the synchronization error between nodes that are close-by. Thereby, nodes periodically broadcast synchronization beacons to their neighbors. Us-

ing a simple averaging method, nodes strive to agree on a common logical clock with their neighbors. Our theoretical analysis of the clock synchronization algorithm of GTSP showed that the logical clock values will converge. Since GTSP is a fully distributed protocol relying on local information only, it is robust to dynamics such as node failures.

To show the feasibility of our approach in practice, we implemented GTSP in TinyOS and evaluated its performance on different testbeds using the Mica2 and TelosB platforms. Overall, the experimental results indicate that the remaining synchronization error between neighbors is small while the global clock skew remains acceptable. Furthermore, we have demonstrated by experiments that GTSP can improve the synchronization error between neighboring nodes significantly in situations where synchronization protocols operating on a tree structure exhibit large local clock skews.

In Chapter 4, we derived *PulseSync*, a novel clock synchronization algorithm, to optimize global synchronization accuracy. It turned out that hitherto proposed protocols suffer from a global skew at least linear in the diameter. In contrast, it can be shown that PulseSync guarantees with high probability a maximum clock difference of $\mathcal{O}(\sqrt{D})$ at the same communication cost as FTSP, the state-of-the-art clock synchronization protocol for wireless sensor networks. The key idea behind PulseSync is to propagate logical clock values as quickly as possible through the network, rather than forwarding an estimate of the reference time that is based on beacons of several synchronization rounds.

We evaluated PulseSync by means of extensive simulations and practical experiments on different testbeds. Both simulations and experiments are in sound accordance with a theoretical analysis of the underlying clock synchronization algorithm [80]. In particular, on a line topology of 20 Mica2 sensor nodes PulseSync features maximum and average global skew of less than 40 and 5 microseconds, respectively, which is approximately a factor 5 better than the corresponding values for FTSP.

Apart from its improved synchronization accuracy, PulseSync is able to adapt very quickly to any kind of dynamics, as clock values are immediately propagated through the network, while many current algorithms will need significantly more time to react to changes. Moreover, the accuracy provided does neither depend on the clock drift nor on the frequency of communication, as long as clock rates do not change quickly compared to the time between pulses. This means that—depending on the stability of environmental conditions—time intervals between pulses may be extended notably.

Part II

Node Localization

6

Introduction to Localization

Knowing the precise location of people, vehicles or goods in real-time is becoming ever more important. In the last decade, the growing availability of positioning systems has spawned a market worth hundreds of billions of dollars. Today, positioning functionality is integrated in modern mobile phones, which have become our constant companions in everyday life. Location based services provide information tailored specifically to our current location and help us to find point-of-interests in unfamiliar surroundings. To this end, the mobile phone determines its current position using a localization method, e.g., based on the Global Positioning System (GPS), connectivity information to cell base stations, and the unique identifiers of Wifi access points. Accurate location information is also a basic building block for wireless embedded systems. Instrumenting the physical world with wireless sensors requires nodes to know about the precise location where each individual measurement took place [38], since sensor data without the “when and where” are almost always meaningless. Furthermore, location information may also be leveraged to improve the operation of the wireless network itself, e.g., to decide how a packet should be routed using geographic information [67, 69], or to adapt the transmission power based on the distance to the receiver [85]. Therefore, accurate position information is helpful to save energy which is a scarce resource in battery-powered embedded systems.

Node localization is defined as the estimation of the current position of a node relative to a reference position, a so called *anchor*. Thereby, an an-

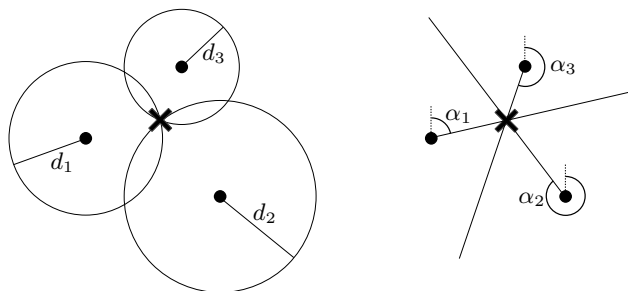


Figure 6.1: Node localization using three anchor nodes: trilateration (left) and triangulation (right). The position of the non-anchor node is indicated by the cross

chor can be any other node or landmark, as long as its location is known. In a two-dimensional Cartesian coordinate system, the position of a node is uniquely determined by its x- and y-coordinates. Using distance or angle measurements from or to an anchor node, the unknown position of a node can be determined as illustrated in Figure 6.1. Acquiring an accurate distance or angle estimation is a challenging problem in wireless networks, since it requires specialized hardware, e.g., directional antennas and measurement of the signal strength. Algorithms to provide node localization have been studied extensively in the context of wireless sensor networks, both in theory [100, 13, 8, 109] and practice [122, 92].

6.1 A Brief History of Navigation Techniques

In the following, we will give a brief overview of the historical development of techniques for navigation in positioning. This section is based on the books “Global Navigation Satellite Systems: Insights into GPS, GLONASS, Galileo, Compass and Others” by Basudeb Bhatta [10] and “The Global Positioning System & Inertial Navigation” by Jay Farrell and Matthew Barth [41].

6.1.1 Dead Reckoning

In the ancient time, navigation was primarily done using *dead reckoning* techniques: The current position is calculated based on an estimation of the spatial displacement and the compass direction from a known position. Such an approach requires precise methods to measure time, speed, and course since the errors due to any imprecise measurements accumulate.

6.1.2 Celestial Navigation

Later in time, travelers and sailors were relying on *celestial navigation*, which depends on the observation of the angle to the sun, the moon or stars to estimate the current position of the observer. The position above the Earth's surface of such a reference object can be determined at any point in time based on a so called *almanac*, a collection of tables containing predicted positions of celestial bodies for that particular year.

6.1.3 Radio Navigation

Further technological advances made it possible to employ electromagnetic waves for navigational purposes starting around 1930. Implementation of *radio navigation* systems were primarily used for instrument flying or landing systems for aircrafts, e.g., *Lorenz*, *VOR*, and *GEE*.

The *Long Range Navigation (LORAN)* system leverages the time difference of arrival between radio transmissions from a chain of spatially separated, synchronized senders to determine the location for aircraft or maritime navigation. A transmission is initiated by the master of the chain followed by the transmission of the slave stations which are delayed for a fixed time interval. The position of a receiver can be calculated using multilateration, which allows to achieve a precision of up to 10 meters using modern equipment.

6.1.4 Satellite Navigation

The idea to use satellites for navigational purposes emerged in the late 1950s after the Soviet Union launched the first *Sputnik* satellite into the Earth's orbit in 1957. First approaches were based on the measurements of Doppler shifts in a signal broadcasted by a satellite (*Transit*, 1962). Later, the satellites were equipped with accurately synchronized clocks, and the positioning was based on the propagation times of the signal emitted by multiple satellites (*Timation*, 1967). First satellites used quartz clocks, which were later replaced by more accurate Caesium atomic clocks. Its successor, the *NAVSTAR Global Positioning System (GPS)* was fully operational with a constellation of 24 satellites in 1993. GPS receivers provide an accuracy of a few meters, which can be improved by deploying additional ground-based reference stations (differential GPS).

6.2 Localization in Wireless Embedded Systems

Node localization and tracking of mobile objects has been studied extensively in the context of wireless sensor networks during the last decade. Accurate position information is mandatory for many sensor network applications. For

instance, environmental or building monitoring rely on carefully placed sensor nodes to gain meaningful measurement data. In other scenarios nodes are mobile, e.g., the sensor is attached to a moving object, or the position is not known a priori, for instance in ad hoc networks deployed for disaster recovery. Furthermore, an object of interest can be located by tracking acoustic or electromagnetic waves emitted by the object itself. In this section, we compare different approaches for node localization or object tracking in wireless sensor networks.

6.2.1 Optical Localization

The *Spotlight* system [142] employs an aerial vehicle, e.g., a helicopter, to generate light events that can be detected by the photosensor device on the nodes. The timestamp of an event detection is used to determine the position of the node. In the *StarDust* [143] system, this approach is extended by attaching optical retro-reflective devices to each node. To localize nodes based on the spatiotemporal properties of an event, the aerial vehicle captures images immediately before and while a light strobe is emitted. In the *Lighthouse* system [119], anchor nodes equipped with a rotating laser emit a beam, which can be observed by nodes using a simple photodiode circuit. Assuming a parallel beam of light, the distance of a node from the lighthouse can be calculated based on the measurement of the total time for a complete rotation and the time during which the node can observe the light flash.

6.2.2 Global Positioning System

The use of receivers for the Global Positioning System (GPS), the state-of-the-art localization solution, is often not applicable for sensor networks due to several reasons: One of the main limitations of GPS is the requirement of line-of-sight to the GPS satellites, which is not given in *indoor* deployments, or in the presence of obstacles. However, many sensor networks will be deployed indoors, e.g., as a part of building monitoring, health care, or inventory management applications. Moreover, operating a GPS receiver accounts for a large fraction of the power budget of a sensor node. Duty-cycling the GPS receiver can lead to significant energy saving. However, there exists a tradeoff between the duty-cycle of the GPS receiver and the uncertainty in the estimation of the node position [64].

6.2.3 Radio-based Localization

A rich body of prior work proposed to employ the integrated radio transceiver of sensor nodes for ranging and positioning applications. Range-free methods rely on connectivity information only, e.g., [14, 133, 103, 52, 163], while range-based methods employ the radio signal strength for localization, e.g., [5, 160,

155]. However, it has been shown that the received signal strength (RSS) is a bad indicator for the distance, due to the effects of multi-path propagation, hardware differences, and antenna characteristics, e.g., [105, 140].

Besides using information about connectivity and signal strength, several other approaches have been applied for node localization in sensor networks. Measurement of the propagation time or the time difference of arrival of electromagnetic waves requires specialized hardware, and are thus often impractical for sensor nodes. Radio interferometric location is based on the superposition of two radio waves, transmitted at slightly different frequencies by two senders. The relative phase offset of the radio signal at two receivers can be used to calculate the distance between the nodes [92]. However, the application of radio interferometry requires high node density in the network. RF Doppler shifts measured on the Mica2 platform have been used to track mobile nodes [73].

6.2.4 Acoustic Source Localization

Various different platforms for acoustic source localization applications have been proposed. The *ENSB*ox [47] is a distributed localization and processing platform used for example in habitat monitoring [3]. It features a microphone array for acoustic source localization and is built around a powerful ARM/Linux core, which allows for sophisticated signal processing. Furthermore, the platform can be self-calibrated using wide-band chirp signals and digital signal processing. The countersniper system proposed in [135] is able to detect the location of a sniper in urban terrain. A gunshot produces a short, easily distinguishable pulse, which is detected by an array of microphones. The hardware is optimized for high-frequency signal processing. Nodes need highly synchronized clocks to estimate the position of the sniper accurately. While acoustic localization provides precise distance measurements within a few centimeters, its range is usually limited to a few meters. Using more powerful speakers to produce sound events at known locations, for instance in the form of an artificial thunder, the time difference of arrival between the acoustic wave and a radio packet transmitted simultaneously can be used to calculate the position of nodes using trilateration [162].

6.2.5 Ultrasound Localization

Ultrasound is a common technique to measure distances and to locate objects or people. A major advantage compared to acoustic positioning is the operation outside the range of human hearing, which allows to use ultrasound for “stealth” applications. Furthermore, ultrasound propagates at the speed of sound (~ 340 m/s at sea level), which is orders of magnitude slower than the speed of light. Small differences in the time of arrival can thus be measured with clocks providing an accuracy in the order of microseconds. The *Bat*

indoor location system [54] uses active ultrasound tags, the so-called bats, which are attached to an object of interest or a person. Ultrasound receivers are mounted on the ceiling, measuring the time of flight of an ultrasound pulse emitted by a bat. Three-dimensional *trilateration* is used to determine the position of the corresponding tag.

The *Cricket* platform [113] has introduced ultrasound ranging into the field of wireless sensor networks. It combines a low-power sensor node with an ultrasound transmitter and receiver. Cricket exploits the substantial difference in propagation speeds between radio and ultrasound signals to measure the distance between two nodes. Having only a single transmitter/receiver pair, ranging capabilities of the Cricket platform are limited to one sector. Commonly, and similarly to the Bat system, beacon nodes are mounted on the ceiling to track listeners deployed on the floor. This setup allows to track mobile nodes in an indoor environment with an accuracy of a few centimeters. The *Cricket compass* [115] uses fixed active beacons placed at known locations and multiple passive ultrasound receivers to determine position and orientation of the node. The *Calamari* platform [154] features a reflective cone on top of the ultrasound transceiver, yielding an omnidirectional beam pattern at the cost of a reduced range. *Medusa* [123] uses an extension board containing an array of four ultrasound receiver and transmitter pairs to cover the hemisphere above the node.

7

Localization with Ultrasound

Having access to accurate location information is a key requirement for almost any wireless sensor network application. Ultrasound based ranging systems employ sound pressure at frequencies above the human hearing range. A main advantage of ultrasound, compared to GPS and other alternative techniques, is its accuracy, providing distance precision in the order of a few millimeters. The location of a node is estimated using distances measured to a few neighbors. If nodes live in a plane, having accurate distance information to two neighbors is still not enough to position a node. Potentially, nodes can even iteratively figure out their location, starting with only a few anchor nodes that know their position. However, in order to achieve iterative positioning, the density of the network must be high, e.g., to allow for rigidity arguments [99]. Unfortunately, ultrasound technology is limited in range, and hence, high node density is required. Furthermore, ultrasound sensors exhibit a limited beam angle. Therefore, the ranging capability suffers substantially at an increasing angle offset between transmitter and receiver. Another issue is that ultrasound distance measurements only work reliably if nodes are in line-of-sight. In this chapter, we present the SpiderBat positioning system, which tackles the aforementioned issues of ultrasound ranging. We start by describing the design and implementation of our approach, followed by an experimental evaluation of the performance of our platform both in a controlled setting and in real-world scenarios. The work presented in this chapter is based on [104].

7.1 The SpiderBat Platform

SpiderBat is designed as an extension board that can provide ultrasound ranging capabilities for existing sensor node platforms, e.g., the Crossbow TelosB and IRIS motes. The basic architecture of SpiderBat and its integration with the host node is depicted in Figure 7.1. The core of SpiderBat is a dedicated microcontroller that is used to control the transmit operation and to process the received signals. The ultrasound ranging board features four independently controllable ultrasound transmitters and four ultrasound receivers, which are placed alternately at the edges of the board, as shown in Figure 7.2. Using multiple transmitters and receivers has several advantages. On the one hand we have an omnidirectional beam pattern, and on the other hand we are able to calculate the angle of arrival. Furthermore, a digital compass provides information about the absolute rotation of the node. The SpiderBat board has to be supplied with power by the sensor node, which allows to switch it off completely when not needed. The microcontroller is connected to the host node through the serial peripheral interface (SPI) and by two interrupt lines. In the remainder of this section, the individual building blocks of the SpiderBat prototype hardware are covered in more detail.

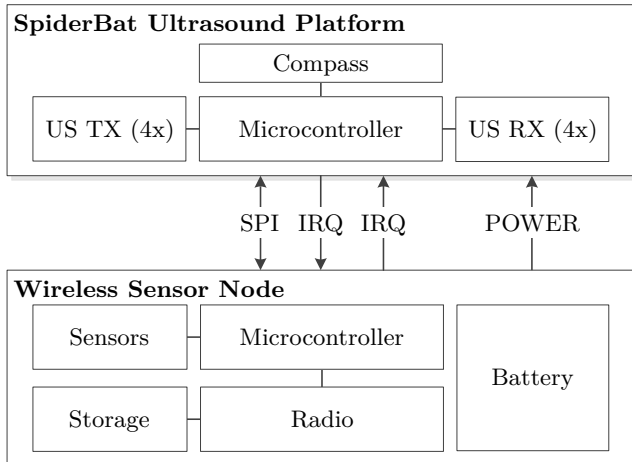


Figure 7.1: The architecture of the SpiderBat system. The ultrasound ranging board is connected to the host node using standard interfaces (SPI, GPIO and Power).

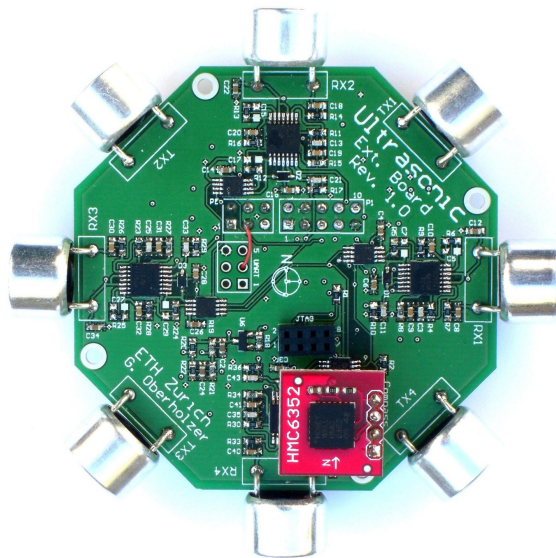


Figure 7.2: Top view of the SpiderBat ultrasound board with a digital compass attached. The board has a diameter of 6.5 centimeters (2.56 inches).

7.1.1 Ultrasound Circuits

The ultrasound transceiver and receiver circuits have been designed with a focus on low complexity and cost. We used off-the-shelf ultrasound transducers with a center frequency of 40 kHz and a bandwidth of 1 kHz. To keep the complexity of transmitter and receiver circuits low, our system relies on the detection of the first peak of an ultrasound signal, rather than performing more sophisticated signal processing (e.g., pulse shaping using a chirp sequence). Furthermore, we decided in favor of separate ultrasound transmitters and receivers, rather than increasing the complexity by switching the circuits between receive and transmit mode. In the following, we highlight some key design aspects of the ultrasound transmitter and receiver circuits.

Ultrasound Transmit Circuit

To maximize the detection range, high output power is needed during the short time interval when the ultrasound transmitters are active (typically 250 μ s). Therefore, a switched DC/DC converter is used to convert the supply voltage of the extension board (typically 3 V) to the operating voltage of the transmitters (12 V). It takes approximately 2 ms until the DC/DC converter

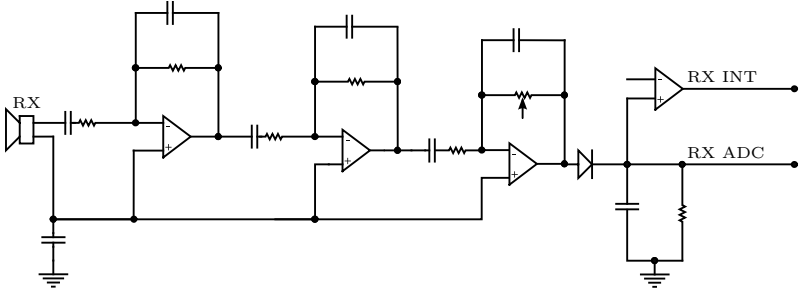


Figure 7.3: Schematic view of the receiver circuit. The received ultrasound signal is amplified and fed to an analog-digital-converter input of the microcontroller (RX ADC). The comparator output triggers an interrupt when the received ultrasound signal exceeds the specified threshold (RX INT).

reaches its nominal output voltage after startup. Since the transmitters require a stable output voltage, large capacitors are deployed upstream of each transmitter for power decoupling. The power supply of the transmitter circuits can be switched off completely to reduce the power consumption when operating the ultrasound board at low duty-cycles, e.g., when ultrasound pulses are transmitted every few seconds only. The microcontroller generating the output signal is operated at a much lower voltage than the ultrasound transmitters. Thus, an operation amplifier with a high supply voltage range is connected upstream of each transmitter. The ceramic ultrasound transmitter is driven at its resonance frequency (40 kHz) by a pulse-width modulation (PWM) output of the microcontroller.

Ultrasound Receive Circuit

Detecting ultrasound pulses reliably and assigning an accurate timestamp to each received pulse is the main technical challenge in systems for ultrasound localization. Since the received signal is typically in the range of a few millivolts only, it needs to be amplified for reliable detection of ultrasound pulses. Therefore, each receiver is connected to three amplification stages, as shown in Figure 7.3. It is necessary to cascade the amplification due to the limited gain-bandwidth product (GBP) of the operation amplifiers. The ultrasound frequency of 40 kHz and the amplifiers having a GBP of 5 MHz result in a maximum gain of 42 dB per amplification stage. In our setup, the first two amplification stages provide an amplification of 21 dB each. The third amplification stage is equipped with a digital potentiometer to adjust the detection threshold and to prevent saturation of the sampled signal. Thus, the overall amplification gain is dynamically adjustable in the range between 58 dB and

75 dB. Finally, the amplified signal is rectified and low-pass filtered. The parameters of the low-pass filter are chosen such that false detections are prevented and the signal rise is not delayed significantly. The amplified receiver signal is connected to an analog-digital-converter (ADC) input pin of the microcontroller, which allows to sample the received ultrasound signal. To unburden the microcontroller from sampling the input signal continuously, a comparator circuit is used to indicate the presence of ultrasound signals. This detection signal is connected to a timer capture input of the microcontroller, which provides a hardware interrupt with an accurate timestamp for the first received ultrasound peak. Thus, our architecture offers the flexibility to choose between low-power operation (peak detection in hardware) and continuous sampling of the input signal for more advanced applications.

7.1.2 Data Processing

The SpiderBat board features a MSP430F2274 low-power microcontroller from TI with 1 kByte RAM and 32 kByte ROM. We decided to include a dedicated microcontroller since most existing sensor node platforms do not provide enough free timers or I/O pins to operate the ultrasound hardware. Furthermore, performing an ultrasound transmit or receive operation is highly time critical. Having this functionality implemented on the sensor node itself would possibly interfere with other time critical operations, e.g., controlling the radio transceiver. The MSP430F2274 provides two hardware timers, each having two independent inputs for time capture and two outputs to generate a PWM signal to control the transmitters. The comparator output of each receiver is connected to a timer input, while the amplified signals of the receivers are connected to the 10-bit ADC input pins. Moreover, we use a Honeywell HMC6352 2-axis digital compass to determine the absolute node orientation. It is connected using a 4-pin socket on the top side of the SpiderBat board. The current heading value from the compass can be read using the I²C bus.

7.1.3 Sensor Node Interface

The SpiderBat ultrasound board can be connected to different sensor node platforms using a 16-pin connector. Power has to be provided by the host node. Table 7.1 reports the measured power consumption for different operation modes of the extension board. The host node (master) controls the ultrasound board (slave) using the SPI bus. Each data transfer is started by a command byte that determines the operation type. The microcontroller on the ultrasound board exposes a register address space for read and write access by the host node. This allows the host node to configure several parameters of the ultrasound ranging operation and to read back the measurement results. Furthermore, two interrupt lines, one for each direction,

Operation mode	Current [mA]
Idle	0.32
Ultrasound RX (min/max gain)	4.68/4.76
Ultrasound TX (peak current)	~100

Table 7.1: Current consumption of the ultrasound board for different modes.

are used for mutual notifications between the host node and the ultrasound board. This unburdens both sides from having to poll for status changes. While the extension board is busy with a receive operation, the host node can remain in low-power state. Once the receive operation is completed, the host node will be notified by an interrupt. Similarly, the microcontroller on the SpiderBat board remains in low-power state when no ultrasound ranging operation is active, otherwise it is woken up by an external interrupt.

7.2 Ultrasound Ranging

An ultrasound ranging operation is always unidirectional. The sender node initiates a single measurement by broadcasting a radio packet followed by an ultrasound pulse, while nearby nodes listen for incoming ultrasound waves.

To measure the time difference of arrival between the radio packet and the ultrasound pulse accurately, both the radio and ultrasound transmissions have to be started concurrently, e.g., as with Cricket [113]. However, in our implementation the ultrasound transmission is started after a fixed time interval following the initiation of the ranging procedure by the sender node. Shifting the start of a ultrasound transmission has several advantages in practice. First, it is not necessary to modify the radio driver to start the ultrasound transmission at the same time as the radio transmission. Depending on the radio chip this might not even be possible since we do not have control over the precise timing. Second, the received radio packet can be processed in the application layer before the ultrasound pulse reaches the receiver. This enables the receiver node to start the ultrasound receivers only when necessary, providing a low duty-cycle operation of the extension board. A similar approach is also used on the Medusa platform [123], where the ultrasound transmission is started when the radio transmission ends.

Figure 7.4 shows the timeline of events during a typical ranging procedure. First, the sender node configures the ultrasound board for a transmit operation. After a certain delay, which is required to ramp up the power supply for the transmission section, the node triggers an interrupt and stores T_{start} , the corresponding local time of this event. Then it broadcasts a radio packet containing a measurement request for all other nodes in the sender's radio communication range. This packet is timestamped at the MAC layer [91] on

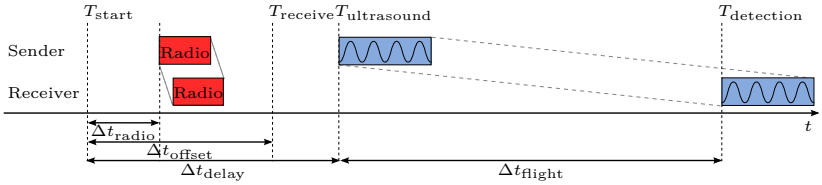


Figure 7.4: Timeline of events during a single ultrasound measurement between a sender and receiver node. The measurement is initiated by a radio packet, which is followed by an ultrasound pulse.

both the sender and receiver, and contains the interval elapsed since T_{start} .

Next, the SpiderBat board starts the ultrasound transmission after Δt_{delay} . This delay has to be larger than the total radio communication delay Δt_{radio} , which highly depends on the delay introduced by the MAC layer protocol. We use $\Delta t_{\text{delay}} = 20$ ms in our implementation.

Nodes that have received the radio packet power up the ultrasound receivers on the extension board. At the same time, an interrupt is triggered to initiate a time of arrival measurement on the extension board and T_{receive} , the local time of this event, is stored. With the help of sender-receiver time synchronization [72], the receiver can determine T_{start} , the start of the transmission in the receiver's local time. Furthermore, the receiver calculates the time difference

$$\Delta t_{\text{offset}} = T_{\text{receive}} - T_{\text{start}} \quad (7.1)$$

that has passed between the start of the ranging procedure on the sender side (T_{start}) and the start of the ultrasound measurement on the receiver side (T_{receive}). When the two nodes are rather close, e.g., less than one meter apart, this delay can exceed the actual time of flight of the ultrasound message. The receiver node is interrupted by the extension board when at least one receiver has detected the ultrasound signal or the receive timeout has expired. Finally, the receiver node is able to calculate the time of flight of the ultrasound pulse as follows:

$$\Delta t_{\text{flight}} = T_{\text{detection}} - T_{\text{receive}} - (\Delta t_{\text{delay}} - \Delta t_{\text{offset}}) \quad (7.2)$$

The corresponding distance d between the sender and receiver node follows directly by multiplication with the propagation speed c of ultrasound:

$$d = c \cdot \Delta t_{\text{flight}} \quad (7.3)$$

7.2.1 Clock Synchronization

Since the radio communication delay is measured using the local clock of the host node, and the time of arrival is calculated at the ultrasound board, it

is important that the clock drift between the two microcontrollers is kept minimal. If possible, they should both be sourced by the same clock to eliminate errors due to clock drift.

Furthermore, the accuracy of a time of arrival measurement depends on the clock granularity. The local hardware clock of the TelosB motes, for example, is sourced by a 32 kHz crystal, which corresponds to a quantization error of roughly 1 cm. Other mote platforms, e.g, the IRIS nodes, have a hardware clock sourced by a crystal quartz, which provides a stable 1 MHz clock.

Since we are only interested in the time difference of arrival and not the absolute arrival time, synchronization of clock offsets between neighboring nodes is not required with our approach. However, clock drift between different nodes affects the accuracy of ultrasound ranging since the delay before transmitting the ultrasound pulse (Δt_{delay}) is measured using the local hardware clock as a reference. Quartz crystals used as clock sources on mote-class hardware exhibit clock drift up to 40 ppm. Thus, when having a worst-case clock drift of 80 ppm between two neighbors, the estimation of Δt_{delay} at the receiver is off by 1.6 clock ticks, which corresponds to a distance of approximately 0.5 mm at 1 MHz clock speed. However, the estimation of the time of flight depends solely on the clock drift of the receiving node. In the worst-case of 40 ppm drift, this contributes roughly two clock ticks to the measurement error at the maximum ranging distance of 15 m. Therefore, not synchronizing local clocks by running a dedicated time synchronization protocol introduces a ranging error of less than a millimeter.

7.2.2 Accuracy of Distance Measurements

In order to evaluate the accuracy of distance measurements with SpiderBat, we used two sensor nodes equipped with an ultrasound board. The receiver node records the timestamp when the peak detection signal is triggered for each ultrasound receiver. Since we were not able to measure ground truth distances within sub-millimeter accuracy, we were mainly interested in the variance of the distance estimation. Since the measurements were performed in an indoor environment, we assumed that the ambient temperature, and thus the speed of sound, remained constant during all measurements.

The standard deviation of the ranging error is 0.31 mm at a distance of 1 m and 5.39 mm at the distance of 14 m, as shown in Figure 7.5.

The absolute error when compared to the distances measured with a measuring tape, is in the order of a few millimeters. The received signal strength of ultrasound pulses decreases with increasing distance from the transmitter. Thus, it becomes more likely that the exact start of the ultrasound pulse is missed since the amplitude of the first few wavelengths is below the threshold value of the comparator. Figure 7.6 shows the distribution of the time of flight measurements for all four ultrasound receivers at a distance of

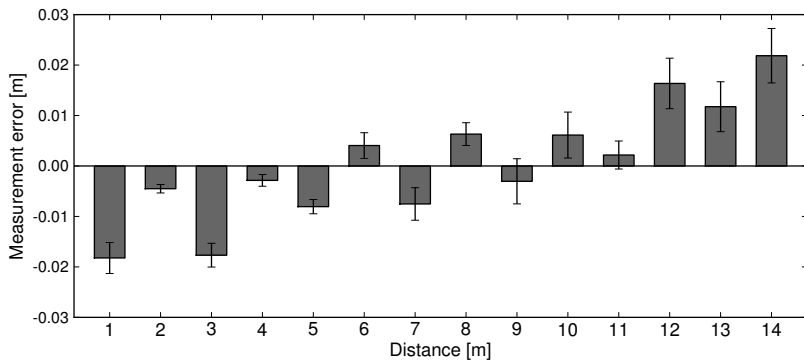


Figure 7.5: Measurement of the ultrasound ranging error at different distances.

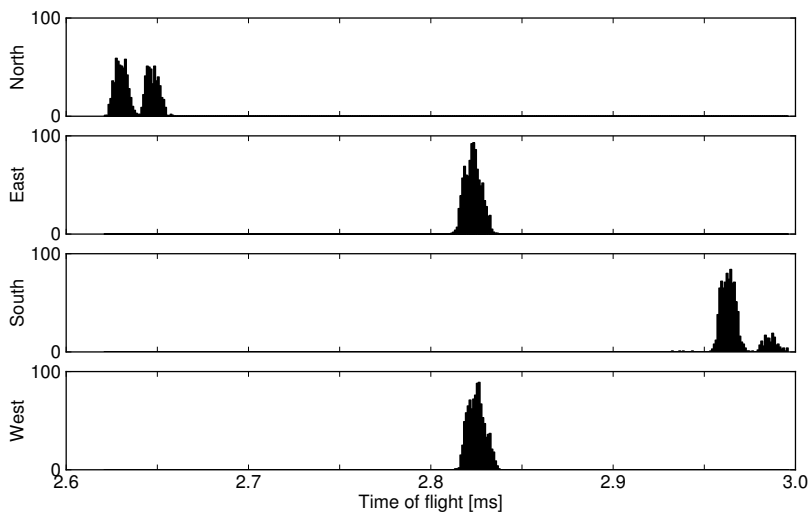


Figure 7.6: Histogram of time of flight measurements for each ultrasound receiver at a distance of 1 m from the sender node.

1 m. Since the northbound receiver points directly towards the transmitter, it detects incoming pulses first, followed by the eastbound and westbound receivers. Eventually, also the southbound receiver detects the pulse. However, we observe that receivers North and South frequently fail to detect the first peak, since the ultrasound signal strength remains slightly below the comparator threshold for the first peak. Instead, they catch a subsequent peak in the signal arriving at a multiple of the wavelength ($25 \mu\text{s}$) later. However, when sampling the ultrasound signal strength immediately after detecting a peak, a strong correlation between the signal strength and the real arrival time of the first peak is observed.

7.3 Angle of Arrival Estimation

The spatial displacement of the different ultrasound receivers on the SpiderBat platform can be leveraged to estimate the angle of arrival of the corresponding ultrasound wave. The relatively slow propagation speed of ultrasound and the board dimensions result in a difference in the time of arrival at the four different receivers, which can be measured using the hardware-based peak detection mechanism described in Section 7.1. In this section, we show how SpiderBat can combine information from multiple ultrasound receivers to estimate the angle of arrival. Furthermore, we evaluate the accuracy of this approach in practice.

The ultrasound receivers on the SpiderBat board have a main beam width of roughly 30° and two side lobes at -45° and $+45^\circ$. Thus, assuming a certain ultrasound receiver is pointing towards 0° , it can cover the area between -45° and $+45^\circ$. However, even a signal arriving from the opposite direction of the receiver's orientation can be detected, but the signal strength is much weaker. If a signal originates from outside this sector, it is received at another receiver first. Consequently, knowing which receiver has detected the pulse first, limits the uncertainty in the angle of arrival to a sector of 90 degrees only.

Furthermore, due to the symmetric shape of the extension board, the arrival time of an ultrasound pulse at different ultrasound receivers can be used to calculate the angle of arrival. Depending on the angle of the incident wave and the signal strength of the ultrasound signal, we can detect a pulse at multiple receivers. Figure 7.7 shows a measurement of the received signal at all four receivers. We can clearly distinguish the detection of the first peak of the ultrasound signal at different receivers.

If at least three receivers have detected the incoming pulse, as shown in Figure 7.8, this results in two mutual differences (Δt_1 and Δt_2) between the time of arrival at the three receivers. By multiplication with the speed of

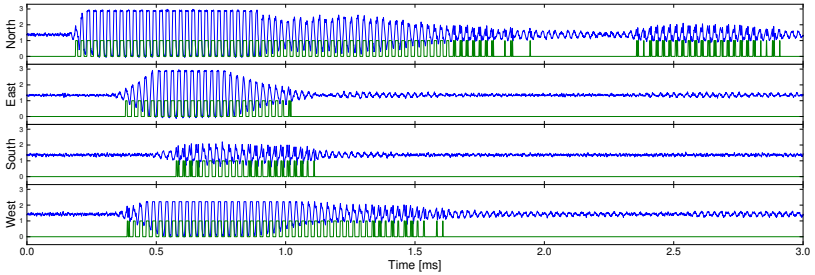


Figure 7.7: Measurement of the amplified ultrasound signal (RX ADC) at all four ultrasound receivers. The direct ultrasound path reaches the receiving node from the north direction, but additional signal paths are visible too. Furthermore, the plot shows the status of the peak detection signal (RX INT) for each receiver.

sound c , we get two corresponding distances l_1 and l_2 as follows:

$$l_1 = c \cdot \Delta t_1 \quad (7.4)$$

$$l_2 = c \cdot \Delta t_2 \quad (7.5)$$

Applying basic trigonometry, we get Equations 7.6 and 7.7, which can be combined into Equation 7.8.

Since the spacing d between the ultrasound receivers is equal, the angle of arrival α of the incoming wave depends on the differences in the time of arrival only.

$$\sin\left(\frac{\pi}{4} - \alpha\right) = \frac{l_1}{d} = \frac{c \cdot \Delta t_1}{d} \quad (7.6)$$

$$\cos\left(\frac{\pi}{4} - \alpha\right) = \frac{l_2}{d} = \frac{c \cdot \Delta t_2}{d} \quad (7.7)$$

$$\alpha = \frac{\pi}{4} - \arctan\left(\frac{\Delta t_1}{\Delta t_2}\right) \quad (7.8)$$

Even if a pulse is detected by only two neighboring receivers, the extension board can still calculate the angle of arrival according to Equations 7.6 and 7.7. Unfortunately, having only a single time difference Δt_1 results in two possible candidates α and α' for the angle of arrival. However, if only two sensors have detected the signal, we can assume that the signal originates in the sector covered by these two sensors. This leads to an unambiguous solution for the angle of arrival α , because otherwise, it is very likely that at least another sensor would have detected the signal too.

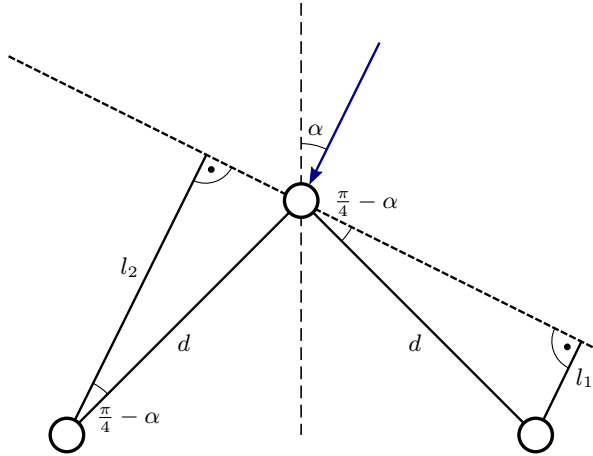


Figure 7.8: Calculation of the angle of arrival α based on three timestamps provides an unambiguous solution.

7.3.1 Distance Correction

The time of flight of ultrasound pulses is always measured between the ultrasound transmitter and receiver pair of minimum distance, as depicted in Figure 7.9. However, this distance has to be corrected with the board radius r in order to position the center of the board correctly. Since the ultrasound hardware is located at the edges of the board, the real distance depends also on the angle between the nodes, on both the transmitter and receiver side. To get an accurate distance between the two nodes, additional angle information is required. Otherwise, an uncertainty in the order of the board radius remains. If the angles α_{rx} and α_{tx} are known, Equations 7.9 and 7.10 can be used to calculate the distance correction term for the transmitter r'_{rx} and receiver r'_{tx} , respectively.

$$r'_{rx} = \begin{cases} r \cdot |\sin(\alpha_{rx})| & \alpha_{rx} \in [\frac{3\pi}{4}, \frac{5\pi}{4}] \cup [\frac{7\pi}{4}, 2\pi] \cup [0, \frac{\pi}{4}] \\ r \cdot |\cos(\alpha_{rx})| & \alpha_{rx} \in [\frac{\pi}{4}, \frac{3\pi}{4}] \cup [\frac{5\pi}{4}, \frac{7\pi}{4}] \end{cases} \quad (7.9)$$

$$r'_{tx} = \begin{cases} r \cdot |\sin(\alpha_{tx} - \frac{\pi}{4})| & \alpha_{tx} \in [\frac{\pi}{2}, \pi] \cup [\frac{3\pi}{2}, 2\pi] \\ r \cdot |\cos(\alpha_{tx} - \frac{\pi}{4})| & \alpha_{tx} \in [0, \frac{\pi}{2}] \cup [\pi, \frac{3\pi}{2}] \end{cases} \quad (7.10)$$

The distance correction term is in the range between 2.8 and 4 cm for the SpiderBat hardware. If we assume uniform distribution of the angle of

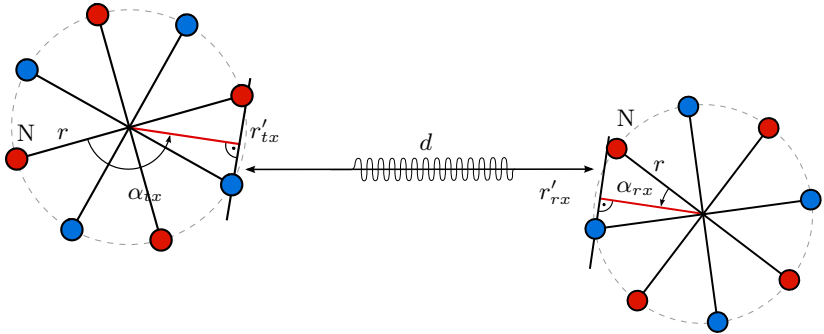


Figure 7.9: Distance measurement between a sender node (left) and a receiver node (right). The distance measurement is performed between the closest transmitter-receiver pair.

arrival, the expected value of the correction term is $r \cdot \frac{\cos(\pi/4)}{\pi/4} = 3.6$ cm, which is the correction term being applied in the absence of angle information, e.g., when a pulse is only detected by a single receiver.

7.3.2 Accuracy of Angle Measurements

We performed a series of measurements to evaluate the accuracy of angle estimation using multiple receivers. Two SpiderBat boards were placed at a fixed distance of one meter such that one ultrasound transmitter points directly towards the receiver node on the other board. During the experiments, the receiving board was rotated step-by-step to measure the time of arrival at different incident angles (0° , 15° , 30° , 45° , 60° , 75° and 90°). Based on the time of arrival information, we calculated an estimation for the current angle of arrival. Figure 7.10 shows the mean and the standard deviation of the calculated angle of the incident ultrasound wave at different angles.

As we can see from the measurement results, the estimation error depends on the particular angle of the incident wave. We observe a measurement error of less than 2° when the incident wave hits the receiver at the main lobe (0° or 90°) or at the peak of its side lobes (45°). In all other cases (15° , 30° , 60° and 75°), the signal strength of the received ultrasound pulse will be significantly lower. Thus, it can happen that the first peak will not trigger the detection signal, but only the second or even third peak will do so. This results in a measurement error in the time of arrival, and thus, also in the estimation of the incident angle.

Clearly, the performance of angle measurements could be improved by adding more ultrasound receivers. However, we observe that by using only four receivers, the mean error in our measurements is below 5° for short

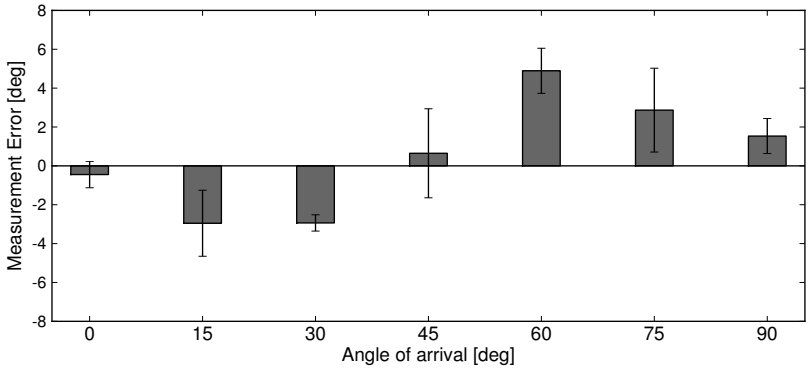


Figure 7.10: Estimation of the angle of arrival using information from multiple receivers. Error bars indicate the standard deviation of the measurement.

distances. Furthermore, the measurements indicate that by applying a non-linear correction function, we can further improve the accuracy of angle measurements with the SpiderBat platform.

7.4 Multi-Hop Positioning

Many existing approaches for positioning assume the presence of one or multiple nodes, so called *beacons* or *anchors*, which have known positions, e.g., [54, 5, 124, 160]. By using iterative or optimization techniques, a positioning algorithm will assign positions to non-anchor nodes. Anchor-free ranging algorithms do not rely on anchor nodes, but obtain position information relative to their neighbors. Range-based localization algorithms utilize distance or angle information acquired by specialized hardware [113, 123, 102, 153, 20], while range-free localization algorithms [133, 103] do utilize connectivity information between nodes only, at the cost of reduced accuracy. In the remainder of this section, we discuss the adaptation of node localization techniques for leveraging distance, angle, and compass information provided by the SpiderBat architecture.

Assuming nodes are positioned in a two-dimensional plane, the position of each node contributes two unknown variables x and y to the localization problem. Hence, to solve the localization problem unambiguously, two linearly independent equations are required for each node. In range-based localization, the set of equations usually consists of the distances between the nodes and several anchor nodes. Relying on distances between nodes only, a simple setup with three anchor nodes, as shown in Figure 7.11, requires three distance measurements to determine the position of a non-anchor node.

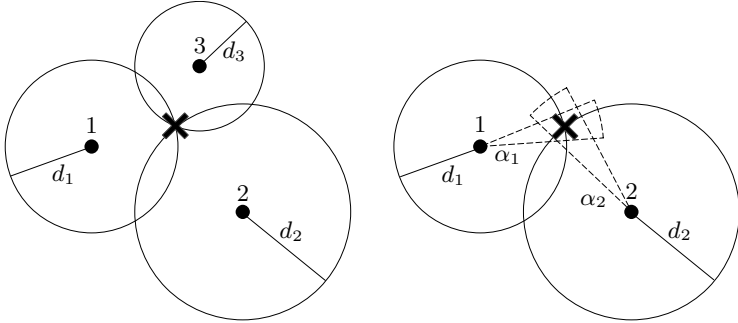


Figure 7.11: Relying on distance information only (left) requires more anchor nodes to position an unknown node (indicated by the cross) than combining distance with angle measurements (right).

However, additional information about the angle towards a not positioned node, as provided by the SpiderBat platform, satisfies to solve the problem unambiguously with only two receivers. Basically, even a single measurement for both distance and angle to an anchor node is enough to determine the position of a not positioned node. However, measurements to additional anchor nodes improve the quality of the positioning. Since the angle of arrival α_{ij} is measured relative to node i 's absolute rotation, we need to convert it into a common coordinate system. This can be accomplished by comparing α_{ij} to the angle α_{ji} of an anchor node j , which is equal up to 180° for line-of-sight ultrasound paths. Thus, we learn the absolute orientation of node i . Alternatively, we can also use the absolute node orientation ϕ_i , provided by the node's compass, to convert relative angles into absolute angles. By comparing the results of these two approaches, we can check the solution for plausibility, e.g., to detect a non line-of-sight path, as discussed later in Section 7.6.

7.4.1 Positioning Algorithm

In the following, we describe a centralized positioning algorithm that employs information about distances and angles. We assume that the algorithm has access to all measurement data (distances and/or angles) gathered at the nodes in the network. This can be achieved by collecting distance and angle measurements of nodes using a data gathering tree rooted at the base station.

The positioning algorithm itself consists of two separate phases. In the first phase, each node is assigned an initial placement within the coordinate system. Then, the node positions are iteratively optimized using the method of least squares.

Initial Node Placement

In the first phase, we assign each not positioned node to the anchor node with the smallest hop distance. This is motivated by the fact that each ultrasound measurement introduces a certain error in the position estimation. Thus, the positioning accuracy decreases with increasing hop count from the anchor node. If two anchor nodes have equal hop count, we further take into account the accumulated distance to the anchor node, since the error in the ultrasound distance estimation increases by distance too. By doing so, we construct several trees, each rooted at the corresponding anchor node. By processing these trees in a top-down manner, we obtain an initial position estimate for each child node j based on the distance d_{ij} and the angle α_{ij} to its parent i in the tree, which is calculated according to Equation 7.11.

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \end{pmatrix} + d_{ij} \cdot \begin{pmatrix} \cos(\alpha_{ij} + \phi_i) \\ \sin(\alpha_{ij} + \phi_i) \end{pmatrix} \quad (7.11)$$

The initial node placement requires the availability of angle and distance information to at least one neighbor. As discussed before, angle measurements have higher requirements than distance measurements since at least two ultrasound receivers need to detect the signal. Otherwise, we only know the distance to the node and in which sector of 90 degrees the signal has originated. In such a case, we need to have another distance measurement to a neighbor to perform the initial placement of the node.

Method of Least Mean Squares

In the first phase of the positioning algorithm, we have not yet taken into account all available measurements between the nodes. In general, distance estimations are usually more accurate than angle estimations and the location error introduced by an inaccurate angle estimation is larger than by an inaccurate distance estimation. Thus, we employ the method of least mean squares (LMS) to improve the location accuracy further by taking into account all available distance information. The squared distance error of node i is given by Equation 7.13, where d_{ij} is the measured distance between node i and j , and \hat{d}_{ij} is the distance according to the current node placement (x_i, y_i) and (x_j, y_j) .

$$\hat{d}_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (7.12)$$

$$e_i = \sum_j (d_{ij} - \hat{d}_{ij})^2 \quad (7.13)$$

The least mean squares method is a steepest gradient descent method, which iteratively reduces the sum of the squared distance errors e_i . Equa-

tions 7.14 and 7.15 show the derivation of the distance error with respect to the x and y coordinate.

$$\frac{\partial e_i}{\partial x_i} = - \sum_j 2 \left(\frac{d_{ij} - \hat{d}_{ij}}{\hat{d}_{ij}} \right) (x_i - x_j) \quad (7.14)$$

$$\frac{\partial e_i}{\partial y_i} = - \sum_j 2 \left(\frac{d_{ij} - \hat{d}_{ij}}{\hat{d}_{ij}} \right) (y_i - y_j) \quad (7.15)$$

The LMS algorithm iteratively updates the current position of the nodes in every step according to Equations 7.16 and 7.17, where $\mu < 1$ denotes the learning rate.

$$x_{i,k+1} = x_{i,k} - \mu \left. \frac{\partial e_i(x, y_{i,k})}{\partial x} \right|_{x=x_{i,k}} \quad (7.16)$$

$$y_{i,k+1} = y_{i,k} - \mu \left. \frac{\partial e_i(x_{i,k}, y)}{\partial y} \right|_{y=y_{i,k}} \quad (7.17)$$

The algorithm terminates after a fixed number of iterations or as soon as the sum of the square errors does not decrease significantly any further, which indicates that a close to optimal positioning for the given distance measurements has been found.

7.5 Experimental Evaluation

In order to evaluate the performance of the SpiderBat platform for positioning in wireless sensor networks, we implemented a localization application in TinyOS. In our prototype implementation, the SpiderBat extension board is connected to a custom node platform with the Atmel ZigBit900 module at its core. The ZigBit module combines an ATmega1281 microcontroller (128 kByte of ROM, 8 kByte of RAM) and a RF212 radio transceiver for the 900 MHz ISM band in a single enclosure. The external quartz of the radio transceiver provides an accurate 1 MHz clock output, which is used as the clock source for the host sensor node and the ultrasound board. The host node and the ultrasound board are powered by two standard AA size rechargeable batteries.

7.5.1 Experimental Setup

We evaluated SpiderBat in an indoor and outdoor setting, as shown in Figure 7.12. The indoor experiments were set up in a gym in order to have all four nodes contained within a large area without any obstacles. We placed



Figure 7.12: Setup of the two testbeds for the experimental evaluation of the SpiderBat platform: The indoor testbed is located in a gym, where four nodes are placed in an area of 10×6 m, approximately 1.5 m above ground (left). The outdoor testbed is located on a sports ground, where we placed four nodes in an area of 10×5 m, approximately 20 cm above ground (right).

four nodes in an area of 10×6 m, approximately 1.5 m above ground. The outdoor testbed is located on a sports ground, where we placed four nodes in an area of 10×5 m, approximately 20 cm above ground. An ultrasound ranging operation, as described in Section 7.2, is initiated by each node once within each measurement round. The receiving nodes report their estimated distance, angle of arrival and compass information to the base station, where the data was collected with a PC. Since all nodes are within the radio communication range of the base station, the use of a collection protocol was not necessary in this setup. After each round, we use the measurement data from all four nodes as input for the localization algorithm.

7.5.2 Experimental Results

During all measurements, Node 1 is an anchor node with a known orientation and position at the origin of the coordinate system. The positions of all other nodes are not known a priori, they are determined using mutual distance and angle information to neighboring nodes. In a first step, we locate the nodes using angle and distance estimates to their closest neighbor only. Next, we apply the LMS algorithm, which takes into account distance estimates between all neighboring nodes. The estimated positions of the nodes are shown in Figure 7.13 and the measurement results are summarized in Table 7.2. The node positions gathered with a measuring tape are depicted in Figure 7.13 for reference.

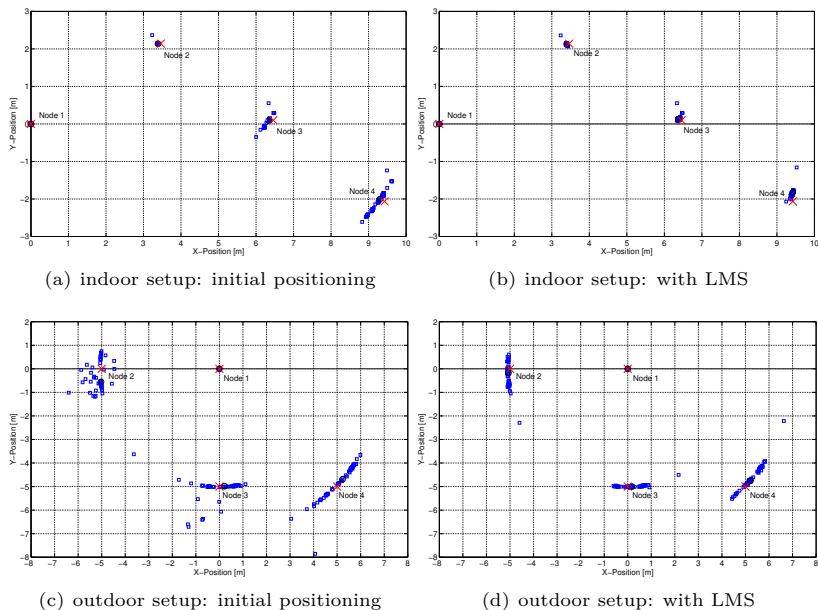


Figure 7.13: Experimental results for the indoor and outdoor setup. A circle indicates the average position over all ultrasound measurements for each node, while the cross indicates the position measured using a reference tape.

Positioning Error (std. dev.) [cm]				
	Node 1	Node 2	Node 3	Node 4
Indoor Setup				
initial positioning	–	2.0	8.3	15.5
LMS method	–	2.2	4.2	5.7
Outdoor Setup				
initial positioning	–	58.2	51.5	61.2
LMS method	–	37.7	36.2	51.6

Table 7.2: Summary of measurement results for the indoor and outdoor setup.

Indoor Experiments

We observed that nodes have both distance and angular measurement only to nodes in close proximity, e.g., Node 2 can measure both angle and distance to Node 1 and 3, but not to Node 4. However, distance information may still be available to nodes that are further away. In the indoor setup, the standard deviation of the localization error is 15.5 cm in the worst-case, which is reduced to 5.7 cm by applying the method of least squares for all known distances between nodes. Furthermore, the measurement results show that the positioning error increases with every hop, as each localization is based on noisy position estimates of the previous node. Although our setup consists of a single anchor node only, we conclude that the SpiderBat platform allows us to localize all three nodes within a few centimeters in an indoor environment. Clearly, adding additional anchor nodes, e.g., employing also Node 4 as an anchor, could further reduce the localization error for non-anchor nodes.

Outdoor Experiments

Although SpiderBat is mainly targeted at indoor applications, we performed an outdoor experiment on a sports ground. The measurement results indicate that the positioning is less accurate compared to the indoor setting. Since distances between nodes are larger than in the indoor experiments, nodes frequently fail to measure an angle of arrival due to the effects described in Section 7.3. Furthermore, ultrasound waves are susceptible to small air disturbances and changes in the ambient temperature, which can impair the measurement accuracy outdoors.

7.6 Non Line-of-Sight Propagation

Ultrasound signals cannot penetrate solid objects, they rather get reflected by them. If the line-of-sight path between two nodes is obstructed, the signal may take a different path including reflections at walls or objects. More generally, a receiver will usually receive the same ultrasound transmission several times. In wireless communication this is a notorious phenomenon, known as multipath propagation. These multipath effects are present in Figure 7.7 already.

In the current implementation of the SpiderBat hardware, an interrupt is triggered when the received ultrasound signal exceeds a certain threshold. As shown in Figure 7.14, sampling the incoming signal during the reception of a pulse reveals valuable information. Due to the limited memory of the microcontroller, only a small amount of samples can be recorded for further processing, e.g., 12 ms of a signal of one particular receiver at a sampling frequency of 80 kHz.

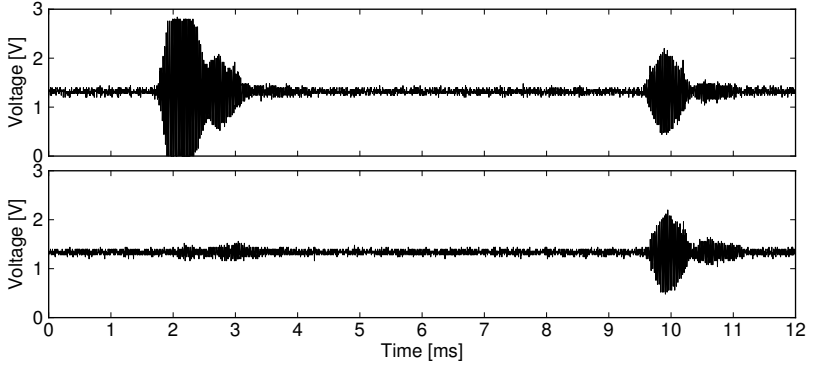


Figure 7.14: Multipath effects measured at a single ultrasound receiver: The line-of-sight signal reaches the receiver after about 1.8 ms. The second peak is a reflection at a near-by wall and is detected at around 9.7 ms (top). Since the line-of-sight path is obstructed in the second measurement, only the reflected path is above the detection threshold (bottom).

As seen in the example of Figure 7.14, indirect paths can be considerably longer than line-of-sight paths. Also, the signal strength of such indirect paths will usually be weaker, and at some point hard to detect. In obstructed environments it is important to detect that two nodes are not in line-of-sight. If not, a positioning algorithm may severely misplace some of the nodes.

Having a dense enough network, with distance measurements between many pairs of nodes, a smart positioning algorithm may detect that indeed some distance measurements will be too large to be possible in Euclidean geometry, for instance because the triangle inequality is violated. It may then ignore these wrong distances and just run the algorithm on the reliable (short) distance measurements.

Thanks to the digital compass and the availability of angle information, SpiderBat has the advantage of *absolute* angle information, which can be used to detect indirect paths more easily. If two nodes are in line-of-sight, they will receive their respective signals at opposite angles, i.e., with an offset of about 180° .

Consequently, we know that two nodes i and j are in line-of-sight if the following equation holds:

$$\phi_i + \alpha'_{ij} = \phi_j + \alpha'_{ji} - \pi \pm \epsilon \quad (7.18)$$

where ϕ_i is the absolute orientation of node i , α'_{ij} is the angle of the signal from node j relative to node i , and ϵ is the total measurement error.

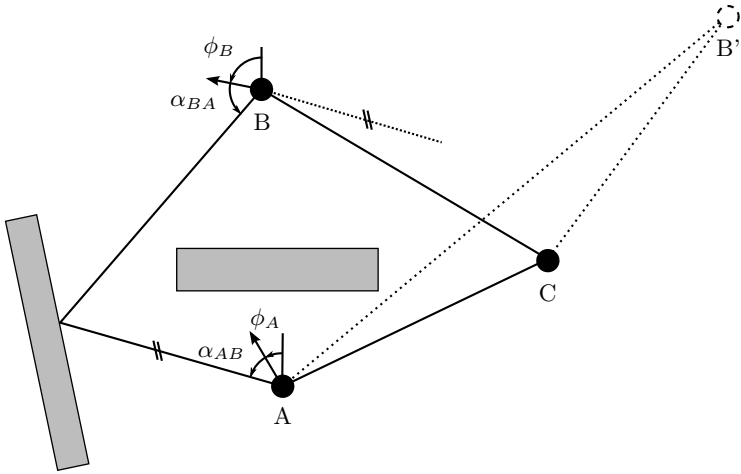


Figure 7.15: Since there is no line-of-sight path between A and B, only the ultrasound signal reflected by a near-by wall can be received. Thus, a generic trilateration positioning algorithm will erroneously confer that B is located at position B'.

On most non-trivial indirect paths, such as the one in Figure 7.15, this is not the case. However, not all indirect paths can be identified using this approach. For instance, if a signal gets reflected at two parallel walls, Equation 7.18 will still be fulfilled. On the other hand, at the price of a higher node density, we may not need a digital compass. For instance, when having three nodes, each node measures the angle between its two neighbors (of the two possible angles we choose the smaller one). Then, if all pairs of nodes are in line-of-sight, the three measured angles should sum up to 180° . If they deviate drastically, we can conclude that at least two nodes are not in line-of-sight.

8

Conclusions

The ability to determine the position of individual sensor nodes is a key requirement for many sensor network applications. While there exist various approaches for providing accurate distance or angle measurements relative to reference positions, there is no obvious choice that fits for the wide range of possible applications of wireless embedded systems. Ultimately, we envision a localization system that is precise, cheap, energy-efficient, and has small form factor. Until this vision can be realized, we have to assess our requirements along the dimensions of the design space of wireless sensor networks. For instance, integrating a GPS receiver in every node may be too much effort for environmental monitoring applications, but mandatory for other purposes.

In this part of the thesis, we contributed a novel platform for node localization in wireless sensor networks, which is targeted to provide positions with an accuracy of a few centimeters for mote-scale devices with limited power budget and processing power. To this end, we presented the design and implementation of *SpiderBat*, a hardware platform for ultrasound ranging applications. SpiderBat is designed as an extension board for wireless sensor nodes, with a focus on the low computational complexity of sensor nodes, low total costs, and with energy efficient operation in mind. The ultrasound transmitters and receivers on the extension board are only active during an ultrasound ranging operation, which has a duration of a few milliseconds only.

Multiple ultrasound receivers and transmitters allow SpiderBat to mea-

sure distances and angles between nodes accurately. Our experiments have shown that the distance between two nodes can be determined in the range of a few millimeters in a controlled environment, whereas the accuracy of the angle of arrival lies within a few degrees, depending on the node orientation and the distance between the two nodes. We evaluated the positioning accuracy of a prototype system consisting of four nodes in both an indoor and outdoor setting. While node localization based on ultrasound pulses provides very high accuracy, the maximal operating range is limited to 15 meters with the current hardware setup.

The combination of angle of arrival and orientation provided by an on-board digital compass provides several advantages over existing platforms: With such a system, we can accurately position nodes, even in sparse networks, where existing techniques fail. In the most simple case, only one anchor node is required to position another node. Furthermore, the absolute node orientation provided by a digital compass enables us to recognize obstacles in the line-of-sight path between nodes.

Besides its application in the classical node localization problem, SpiderBat may also be used to learn about the environment of a node by transmitting an ultrasound pulse and analyzing the signal reflected at nearby walls, obstacles or moving objects. Apart from the technical challenges to implement such an echo application, some other difficulties have to be mastered before one can correctly position walls and obstacles. For instance, objects need to have a certain dimension in order to be visible from multiple vantage points. At this stage, detecting many small obstacles seems beyond the possibilities of the current implementation of the SpiderBat approach. However, in the near future, one may hope of a network of self-organized wireless nodes, e.g., deployed by robots in unknown terrain or buildings, which learn about their environment utilizing ultrasound pulses.

Part III

Applications

9

Resource-Oriented Architectures for Sensor Networks

Early deployments of sensor networks followed a rather monolithic approach, where the software and hardware architecture of a sensor network is tailored for a single purpose, e.g., deployments for habitat [145], structural [65, 19] and environmental monitoring [146, 150, 9]. Recent technological advances gave rise to more powerful embedded platforms, featuring increased storage and processing capabilities. This progress allows to implement protocols and applications with an increased requirement towards memory capacity and communication bandwidth on wireless embedded systems. By creating the *Internet of Things*, physical objects are represented in the digital world. Using unique identifiers, embedded devices can be addressed from the Internet to form a large-scale network of sensors or actuators. Furthermore, users start to interact directly with *smart things* in their proximity or built into their homes by using their mobile phones. Clearly, a system architecture largely different from what was used in early wireless sensor networks is necessary to meet the requirements of such heterogeneous networks. For instance, the network should integrate additional devices and services without the need for manual configuration or reprogramming.

In this chapter, we present an approach to integrate tiny, low-power wireless sensor or actuator nodes into a network based on the *Internet Protocol (IP)* to simplify data access and discovery of services. Sensors and actuators

are represented as resources of the corresponding node and are made accessible using lightweight web services. While many existing approaches provide web services at a smart gateway, we show that it is feasible to provide web services at each node, even when using a very resource-constrained hardware platform. We present a prototype implementation on a mote-class hardware and evaluate the performance of our approach. As a typical application of such as system, we present the architecture of a monitoring and control system for smart buildings in Section 9.4. The work presented in this chapter is based on [131].

9.1 Related Work

Connecting wireless sensor nodes to the Internet has been widely explored in the last few years. Sensor networks running proprietary communication protocols like ZigBee need an application layer gateway to connect to other networks. As devices need to be accessible using standard protocols, implementing IP on the sensor nodes seems to be more suited [25, 31]. The first IPv4-compatible network stack targeting 8-bit microcontrollers has been presented in 2003 by Adam Dunkels [30]. However, the increasing demand for IP addresses initiated by the vision of ubiquitous sensor nodes can be better handled with IPv6. 6LoWPAN [61, 60] is a standard for implementing IPv6 on wireless sensor networks. The integration of *Extensible Markup Language (XML)* based web services into embedded devices has been shown for an ARM7 microcontroller with 256 KBytes of ROM and 32 KBytes of RAM [55]. Priyantha et al. [114] focused on an efficient implementation of web services on sensor nodes by reducing the power-consumption and minimizing the overhead introduced by the transport layer while using XML and the *Web Services Description Language (WSDL)* data formats. Addressing sensor nodes as resources has been implemented for the Sun SPOT platform, which has a 32-bit processor and 512 KBytes of RAM [49]. We use similar ideas for accessing sensor nodes using HTTP methods and coding the answer as JSON objects, but we target mote class devices with an order of magnitude less memory and more severe energy constraints. Furthermore, we use a full IP stack running on our sensor nodes, while a proxy server is used to bridge the requests to the devices in prior work [49].

Approaches to use sensor nodes to reduce the power consumptions in buildings are presented in [49, 114]. For instance, the ACme project [62] uses sensor devices with an energy meter to report the current power consumption to a central server. Thereby, sensor data are presented using the *Simple Measurement and Actuation Profile (SMAP)*, which uses lightweight web services to unify access to measurement instruments and other information providers [27].

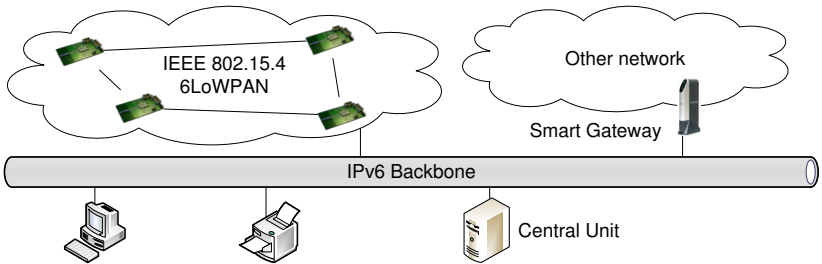


Figure 9.1: Architecture of a monitoring and control system for smart buildings. 6LoWPAN-enabled wireless sensor nodes are directly integrated into the IPv6 network (left). Access to non IP-enabled devices is provided through a smart gateway at the network boundary (right). Sensors and actuators are accessible by a central unit.

9.2 System Architecture

While standard IP-based protocols introduce some overhead compared to a customized protocol, the increased possibilities to connect different devices certainly pay off. Although we are using tiny devices, we still want to use the standard protocols that made the Internet so powerful. It is even possible to run proprietary protocols in certain parts of the network. A smart gateway at the network boundary is used to provide access for IP-based protocols. As a large number of devices may be placed at different locations, for instance throughout a large building, connecting them using wires, e.g., by Ethernet, is often not practical. The IEEE 802.11 wireless LAN standard solves the problem of laying wires, but the energy consumption of the radio transceiver makes it infeasible to operate devices on batteries. We address this problem by employing a wireless sensor network based on the IEEE 802.15.4 physical layer standard, which is optimized for energy efficient communication with low data rates. The 6LoWPAN [98] header compression scheme allows to efficiently send IPv6 packets over IEEE 802.15.4-based networks. The overall architecture of the system presented in this work is outlined in Figure 9.1.

9.2.1 Configuration and Service Discovery

Auto-configuration of network nodes is a mandatory feature when managing large networks. We employ the stateless address auto-configuration mechanisms provided by the IPv6 protocol. After startup, a node sends a *router solicitation* request to the link-local multicast address. A router will respond with a *router advertisement* message containing configuration parameters for the current network. Otherwise, a sensor node can obtain its IPv6 address by

issuing a *Dynamic Host Configuration Protocol (DHCP)* request. Once the address configuration phase has been completed, a node starts to advertise its offered services to the other nodes in the network. We use the *Multicast Domain Name System (mDNS)* protocol, which uses special DNS resource records to provide information about services offered by a device such as the service type, domain name and additional configuration parameters.¹ A node can query a certain service type by sending a DNS packet to the link-local multicast address. Nodes that offer services of the corresponding type will send an answer. Many existing network peripherals implement the mDNS protocol, e.g., printers, scanners, and network attached storage devices.

9.2.2 Web Services for Wireless Sensor Nodes

Web services allow the interaction between different devices over the network in order to exchange data or to trigger certain actions. Data are exchanged between peers using the *Hypertext Transfer Protocol (HTTP)*. Traditional enterprise solutions tend to use the *Simple Object Access Protocol (SOAP)* together with XML for the data representation. Recently, a more lightweight solution called *Representational State Transfer (REST)* has been proposed, which is built on top of the GET, POST, PUT and DELETE methods of HTTP [43]. Web services following these principles are denoted as *RESTful web services* in the remainder of this chapter.

Data Access Scheme

Sensor nodes are responsible to provide information about their sensors and actuators to the central monitoring server. One possibility to guarantee that the server has up-to-date information is to let the sensor node periodically send status messages to the monitoring server. An alternative approach would be that the server polls the state of the sensors on demand. The first approach is preferable if no actuators are connected to the sensor node. However, the second method offers more flexibility since the same interface can be used to access the data of a sensor, as well as to modify the state of actuators. A second architectural decision affects the device where the web service has to be implemented. Two options are common in literature: either implementing the web service at the gateway, which connects the sensor nodes with the Internet, or implementing it directly on web-enabled devices. The second option offers more flexibility since no changes to the gateway functionality are required when deploying new device types.

¹<http://tools.ietf.org/id/draft-cheshire-dnsext-multicastdns-06.txt>

RESTful Programming Interface for Sensor Nodes

Our approach implements a RESTful Application Programming Interface (API) on sensor nodes to provide access to sensors and actuators from an IP-based network. As each sensor node may connect with different sensors and actuators, manual configuration of a central monitoring system is infeasible if a large number of sensor nodes are used. Based on the RESTful API, we introduce a plug-and-play approach, which enables not only the automatic discovery of sensor nodes in a wireless network, but also of the functionality they provide.

The Representational State Transfer (REST) protocol [43] can be outlined as a collection of network architecture principles and is used by various web based applications to expose their functionality by providing an API. Unlike other web services, such as SOAP, which rely on other application layer protocols, REST uses HTTP as the transfer protocol. The functionality of a system is implemented as a set of resources that can be identified using the corresponding *Uniform Resource Identifier (URI)*; thus both the sensor node itself and every connected sensor get their own *Uniform Resource Locator (URL)*. A RESTful web service is a collection of resources. By using the four basic operations provided by the HTTP protocol (**GET**, **PUT**, **POST** and **DELETE**), clients can interact with the resources. Each operation executes a corresponding action like updating a resource or creating a new entry in a collection. Its lightweight stack simplifies the integration of REST in devices with limited resources such as sensor nodes [108, 50]. Not only the device is a resource, but all sensors itself are represented as a distinctive resource that can be accessed over the network. As web services are not limited to sensor nodes, they can be used in any devices integrated in the monitoring and controlling system of a smart building.

Resource Discovery and Access

As an individual sensor node potentially offers many different resources, the device has to provide functionality for automatic discovery of resources. Therefore, the RESTful API provides a modular structure as outlined in Figure 9.2. The API differentiates between three types of resources: *root collection*, *collection* and *member*. The root collection offers a list of all collections the sensor node provides, the collection can be addressed to get a list of all its members while the actual functionality of a sensor or actuator is implemented as a member resource. This modular design enables another device, such as a central unit, to auto-discover the functionality offered by a sensor node. As the server can only request the information it is interested in, the message sizes are kept small, and the number of messages is reduced. The response to a **GET** operation is presented in the JavaScript Object Notation (JSON), which is a lightweight and platform independent data exchange

/	(root collection)
/temperature/*	(collection)
/temperature/celsius	(member)
/temperature/fahrenheit	(member)
/humidity/*	(collection)
/humidity/relative	(member)
/humidity/absolute	(member)

Figure 9.2: Example for the structure of resources provided by a RESTful web service on a sensor node with an integrated temperature and humidity sensor.

format. Parsing and creating the JSON representation of data require less overhead on resource constrained sensor nodes compared to the XML format.

9.3 Implementation

In this section, we describe the implementation of a RESTful communication protocol for wireless sensor networks. While the proposed API can easily be implemented on systems such as PCs, printers or servers, various limitations have to be considered for the implementation on resource constrained wireless sensor platforms.

9.3.1 Hardware Platform

The hardware platform used for the implementation of the proposed communication protocol is a custom design based on the ZigBit900 MCU wireless modules from Atmel. The ZigBit module features an ATmega1281 microcontroller and integrated radio module (Atmel ATRF212) integrated into a small form factor. The ATmega1281 microcontroller comes with 8 KBytes of RAM and 128 KBytes of program ROM. The RF212 radio module is compatible with the IEEE 802.15.4 standard for low data rate communication. We built the *Pixie* prototype platform, shown in Figure 9.3, which is based on the ZigBit900 module and provides connectors for different peripheral devices. The RF signal can be connected to the on-board chip antenna or to an external antenna. The small form factor and the various possibilities to connect additional sensors or other peripherals make the *Pixie* nodes an ideal prototyping platform for wireless sensor network applications. Based on the prospective location of the sensor node in a deployment, one might attach different sensors or actuators to the node. The *Pixie* platform provides extension headers to connect additional components through I/O pins

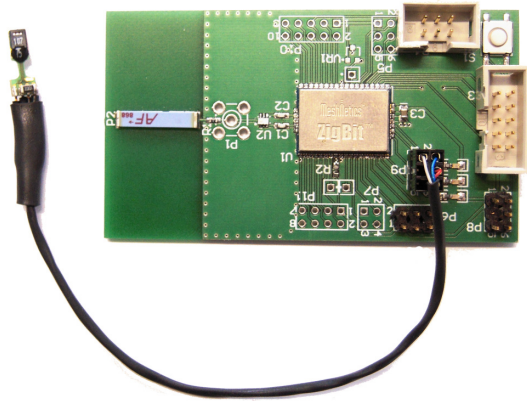


Figure 9.3: The Pixie node is a wireless platform developed at ETH Zurich. It is equipped with extension headers for external peripherals. In this picture, a Sensirion SHT75 temperature/humidity sensor is attached to the node.

or peripheral interfaces (I2C, SPI and UART). This extensibility in combination with the resource discovery mechanism presented in Section 9.2.2 makes the Pixie nodes a convenient platform for prototyping sensor network applications.

9.3.2 TinyOS Implementation

In this section, we discuss how the RESTful API can be efficiently implemented targeting sensor nodes having only 8 KBytes of RAM and running on batteries. We ported the ZigBit module to the TinyOS operating system, which required only small modifications due to the similarity of the ZigBit module to existing platforms. Implementing web services on sensor nodes introduces additional layers in the protocol stack. In the following, we address the overhead introduced by each of these layers, and discuss approaches to implement the functionality given the resource constraints of low-power sensor node platforms.

Internet Protocol (IP)

In order that other devices can directly connect to the sensor nodes, they have to be IP-addressable. As the transmission of IPv6 packets over an IEEE 802.15.4 network introduces a large overhead in the message length, further compression is necessary. The 6LoWPAN standard addresses this problem by introducing an adaptation layer to compress the IPv6 header

and to fragment IPv6 packets in order to enable IPv6 communication in WSNs [61, 60]. Our implementation is based on *blip*, the 6LoWPAN stack included in TinyOS.

Transmission Control Protocol (TCP)

We use TCP for establishing reliable data connections between the sensor nodes and the central unit. Various aspects to reduce the overhead introduced by using TCP as transport protocol are discussed in [114]. In particular, by using persistent TCP connections, the latencies of a request can be drastically reduced. As a typical response of a sensor node consists only of two or three data packets, the establishing and closing of a TCP connection provides a larger overhead. As the central unit regularly connects to the sensor node, persistent TCP connections reduce the number of packet transmissions.

Service Advertisement

We implemented service discovery according to the mDNS proposal. Each node periodically sends a *User Datagram Protocol (UDP)* packet to the IPv6 link-local multicast address to announce the presence of its RESTful web service in the network. We introduced the new service type “`_rest-sensor._ws._tcp`” for the RESTful web service on the sensor nodes. To reduce the complexity of the implementation on the sensor nodes, we only send out announcements; we omit the functionality for processing incoming mDNS packets from other hosts. Since mDNS packets are sent to the link-local address, they are not automatically forwarded in a wireless multi-hop network. Therefore, intermediate nodes have to relay incoming mDNS packets towards the border router.

Web Services Implementation

The final goal we are pursuing is to access the sensors and actuators of a low-power device using a high-level API. For this purpose, we implemented the RESTful API described in Section 9.2 on sensor nodes using the 6LoWPAN communication stack described above. Following the layering outlined in Figure 9.4, the API has been implemented in TinyOS.

Targeting the Pixie nodes having only a small amount of RAM, we implemented a lightweight HTTP layer serving basic tasks of a web server. This layer is the link between the communication stack and the RESTful API. The web server forwards incoming requests to the REST parser and provides the basic functionality to generate HTTP packets. The key component of our implementation is the REST layer, which dispatches incoming requests to the handler method of the corresponding collection, thus mapping a resource

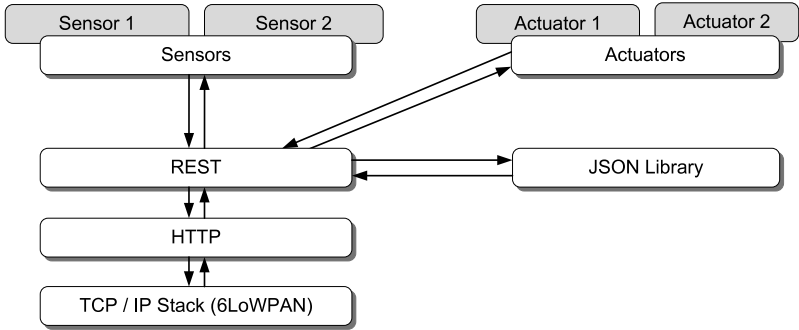


Figure 9.4: TinyOS implementation of the RESTful API. The HTTP and REST layer on top of the TCP/IP communication stack provide the RESTful API with help of an additional JSON library.

to an URI. Furthermore, it generates the response to a root collection request. As the manual generation of JSON objects would be a huge overhead and prone to error, an additional library has been developed to support the reformatting of a response as a JSON object.

A sensor or actuator needs only to bind itself with its collection name to the REST layer and to implement an event-handler for receiving a `GET` respectively `PUT` request in order to provide its functionality in the RESTful API. Upon receiving a `GET` request for a resource, the REST layer forwards the request to the related sensor or actuator. A sensor replies to a `GET` request with a JSON object containing one or more parameters as outlined in Figure 9.5. The JSON library helps to create JSON objects by offering services that automatically generate the JSON envelope.

In addition, `PUT` requests are supported to interact with actuators. The corresponding parameters are added to the `PUT` request as raw data lines as the implementation of a JSON parser would generate a large overhead on a sensor node. When receiving a `PUT` request, the actuator needs to parse the parameters and updates its state according to their values.

The TinyOS implementation is fully platform independent. Compiling the RESTful API for the Pixie platform with various sensors and actuators connected to the device, the RAM footprint has a size of 7.6 KBytes while the ROM footprint has a size of 43 KBytes. However, merely 0.7 KBytes of RAM and 2 KBytes of ROM are used by the RESTful API and the mDNS implementation while the rest is mainly used for the TCP and 6LoWPAN implementation.

```
{
  "device": "temperature",           // name of the resource
  "method": [                       // supported methods
    "G"                             // of the resource (GET)
  ],
  "param": [                        // array with all parameters
    {
      "n": "celcius",               // name
      "v": 26,                     // value
      "t": "i",                    // data type (integer)
      "u": 0                        // updatable flag (false)
    }
  ]
}
```

Figure 9.5: Example of a JSON object sent by the RESTful web service on the node in response to a GET request.

9.3.3 Performance Evaluation

In this section, we evaluate the performance of our implementation of web services on tiny sensor devices. We measured the response time for a HTTP request to the web service on the sensor node. During the measurement period, the central unit triggers periodic GET requests to the sensor node. Measurement results are shown in Figure 9.6. Requests that can be answered with a small HTTP payload length have a response time of less than 200 ms. It can be seen that the response time increases almost linearly with increasing payload length. Persistent TCP connections lead to a smaller response time with the drawback of an increased memory usage on the sensor node.

Energy Efficiency

Reducing the energy consumption of the nodes in the sensor network itself is a crucial aspect in our system. Although external power is often available to operate the sensor nodes, for instance in buildings, one might equip nodes with batteries to reduce the cost of the wiring, and to increase the flexibility in the node placement. While the microcontroller of a node can be placed in power-save mode when idle, the radio chip has to be enabled to catch incoming packets. The power consumption of the RF212 radio chip used on the Pixie platform is 19 mA when transmitting, 9 mA in the receive mode and 0.2 μ A in the power-save mode. To prolong the node lifetime, the radio chip has to be operated in the power-save mode as often as possible.

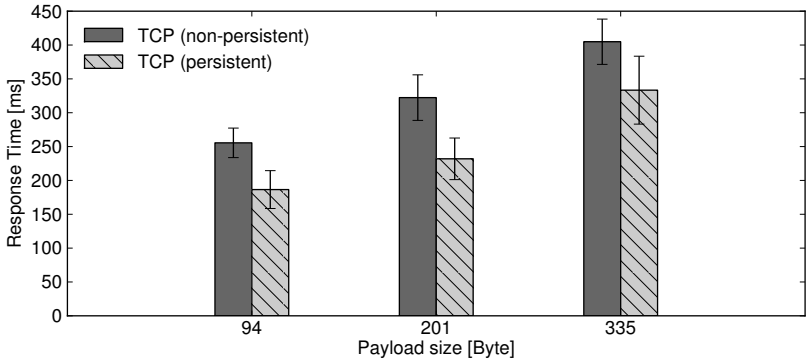


Figure 9.6: Average response times for HTTP requests to the sensor node with and without persistent TCP connections.

Low-power listening reduces the time a node spends for idle listening by duty-cycling the radio chip [111]. A node wakes up periodically from sleep mode to check the radio channel for activity. If there is activity detected, it remains in the receive mode until the packet is received successfully, otherwise it returns back to sleep. The sender node has to transmit a preamble that is at least as long as the sampling interval of the receiving node. The sampling interval and the preamble length can be tuned to meet the desired duty-cycle. However, the per-hop delay increases with an increasing sampling interval. Nevertheless, it can be shown that HTTP requests can be handled with a delay of less than a second in multi-hop networks while utilizing low-power communication protocols [159].

9.4 Applications for Smart Buildings

Improving the energy efficiency of buildings is an important step towards a more sustainable lifestyle, leading to significant cost savings for real-estate owners. Nowadays, energy use in buildings is responsible for roughly 40% of total energy use and for 36% of the CO₂ emissions in the European Union [24]. Recently, the European Commission pledged to cut the annual consumption of primary energy by 20% by 2020.

While recent advances in the field of material science led to an improved energy efficiency of the building envelope, a lot of energy is consumed by different equipment such as HVAC (heating, ventilating, and air-conditioning), lightning, home and office appliances. In a first step, decentralized sensor nodes of different types are required to report the current energy usage or

environmental conditions to a centralized monitoring system. A smart power outlet will report the current energy usage of the attached device to a central server. Temperature sensors are used to react upon variations in room temperature. Actuator nodes are responsible to control small subsystems within the building. A presence sensor can automatically switch off the ceiling lighting when an employee has left the office. However, even more energy can be saved when different subsystems cooperate with each other. For example, the central control system can switch off the office lights, lower the heating and send an employee's PC to standby mode when the door system reports that the employee has left the building.

Building automation and control systems rely on many sensors and actuators placed at different locations throughout a building. Reducing the power consumption of a modern building requires continuous monitoring of various environmental parameters inside and outside the building. The key requirement for an efficient monitoring and controlling is that all sensors and actuators are addressable over the network. Different proprietary protocols and industry standards for building automation have been proposed over the years, e.g., CEBus, EIB, BACnet or Local Control Network (LCN). However, integration of different services based on IP networks has become an important trend during recent years.

As a proof of concept for our approach, we developed a web application to monitor and control the sensor network using the *Google Web Toolkit*.² The goal of the web application running on the central unit is to provide a user-friendly way to get information from the sensor nodes and to control the actuators. The back-end layer communicates with the sensor nodes over the RESTful API described in Section 9.2. To detect new devices plugged in the network, the central unit listens to mDNS messages sent out by the clients. Whenever the web application detects a new device supporting the service type “`_rest-sensor._ws._tcp`”, it automatically discovers the functionality offered by the device. Furthermore, the web application allows to define a set of rules that trigger certain actions based on a specified event. A typical rule may have the form “*if the motion sensor in room B4 does not detect any movement for five minutes, turn off the light in B4 and send the PC to standby mode*”. As the functionality of the devices is discovered by the central unit, the effort required to set up monitoring applications for smart buildings can be significantly reduced. In order to make a device accessible by the web application, it only has to send out mDNS messages and to implement the RESTful API.

In addition, sensor nodes can provide push-based access to sensor data using a simple subscription mechanism based on the RESTful API. Once the central unit or another node subscribes to a data stream, the node periodically transmits sensor readings to the subscriber using UDP packets. By

²<http://code.google.com/webtoolkit/>

The screenshot shows a web interface for managing sensor network resources. On the left, a tree view lists resources under two sensor nodes: 'sensor-101.local' and 'sensor-102.local'. Under 'sensor-101.local', there are 'actuators', 'sensor', 'light', 'report', and 'management'. Under 'sensor-102.local', there are 'actuators', 'sensor', 'report', and 'management'. The 'sensor' resource under 'sensor-101.local' is expanded to show 'temperature' and 'singleValue'. The 'singleValue' resource is selected, and its details are shown in the main panel.

The main panel is titled 'TEMPERATURE' and shows the URL '(http://sensor-101.local/sensor/temperature/singleValue)'. Below the title is a table with the following data:

Parameter	Value	Datatype	Update	Clear
value	3328	Integer		
celcius	26	Integer		

Below the table, there are tabs for 'Full Answer', 'Allowed Methods', 'History', 'Record', 'New Event', and 'Stored Events'. The 'Full Answer' tab is selected, and it displays a JSON object representing the resource:

```
{
  "device": "Temperature",
  "method": [
    "G"
  ],
  "param": [
    {
      "n": "value",
      "v": 3328,
      "t": "i",
      "u": 0
    }
  ]
}
```

Figure 9.7: Screenshot of the web interface provided by the central unit. The set of resources provided by the sensor nodes are displayed in a tree view (left). Once a resource is selected, properties can be viewed and edited in the main panel (right).

employing this two-fold approach, we can combine the flexibility provided by RESTful web services with the smaller overhead of UDP.

10

Personalized Sensing Applications

In the last decade, wireless sensor networks research has been driven by advances in microelectronics and wireless communication. Early deployments of sensor networks, e.g. [145, 146], were characterized by a relatively small number of nodes, carefully placed near an object of interest. Research mainly focused on the networking aspects, rather than on the sensor data itself. In the last years, mobile phones have attracted a growing interest as a platform for people-centric sensing applications [33]. Thanks to the large penetration of mobile phones in all demographic groups and due to the increasing capabilities of integrated sensors, this provides the opportunity to gather data at unprecedented fidelity and scale. While modern devices already offer built-in imaging capabilities, microphone, GPS receiver, accelerometers and gyroscopes, future advances in miniaturization of micro-electromechanical sensor systems will bring further sensing possibilities to mobile phones. Mobile sensing has a great potential to tackle important problems in various application domains including health care, transportation, environmental monitoring, and urban development. In this chapter, we study hardware and software architectures for personalized sensing applications. The work presented in this chapter is based on [39].

10.1 The Mobile Phone as a Sensing Platform

With the mobile phone as a ubiquitous sensing, computing and networking platform in our pockets, a wide range of novel applications become feasible. In the last few years, operating systems such as the Google's Android OS or Apple's iOS have greatly simplified access to the integrated sensors through sophisticated programming interfaces for application developers. The huge success of mobile distribution channels for small software applications allows to reach a large number of potential users with minimal effort. Therefore, people-centric sensing application have received considerable interest by researchers in the last few years. Participatory sensing [15] enables the use of privately owned and controlled mobile devices to perform a collaborative sensing task. Individuals collect sensor readings during work or leisure activities, which allows to gather high fidelity data about the surroundings of participants leveraging the local knowledge of the user [134]. It is even possible that participatory sensing applications are initiated by members of the community itself without the need for a supervising organization, or that mobile phones collect sensor data opportunistically without explicit control by the users [76]. Mobile phones have been employed to gather information about a user's context, e.g., [96, 4]. The use of mobile phones in participatory sensing is manifold. In the Pothole [37] and Nericell [97] projects, mobile phones provide sensor data to measure traffic and road surface conditions. BikeNet [34] uses custom hardware together with a mobile phone to gather information about cyclist performance and environmental conditions. Mobile phones can be employed to generate noise maps in urban areas using the integrated microphone and the location of the user [116].

10.1.1 Interfacing External Sensors

External sensors can extend the capabilities of a mobile phone as a platform for sensor data collection. The microSD card slot available in many phones can be used to connect peripherals using the SD input/output (SDIO) standard. However, the physical location of the slot makes it often infeasible to connect cables or sensors. The Android Open Accessory API grants read/write access and a power supply for peripheral devices over the USB port of the phone. Another approach connects to the headphone jack of the phone to harvest energy and to use the bi-directional audio interface for low-rate data communication [70].

Therefore, several approaches exist to interface external sensors wirelessly to mobile phones. Bluetooth capabilities are integrated in almost any modern phone to connect external peripherals such as headsets. However, a severe drawback of the Bluetooth protocol is its power consumption, which is specifically addressed with the introduction of a low energy profile in Bluetooth version 4. Other low-power protocols such as the IEEE 802.15.4 standard or

Protocol	Module	RX	TX	Sleep
		[mA]	[mA]	[μ A]
Bluetooth 2.1	Roving Networks RN-42	35	65	250
Bluetooth 4.0	Nordic nRF8001	14.5	13	0.5
IEEE 802.11 b/g	Roving Networks RN-171	38	120	4
ANT	Nordic nRF24AP2	17	15	0.5
IEEE 802.15.4	Atmel AT86RF231	12.3	11.6	0.02

Table 10.1: Comparison of the typical current consumption for different radio transceivers and operation modes (source: device datasheets).

ANT/ANT+ have not yet gained widespread adoption by manufacturers of mobile phones. The Wi-Fi chips integrated in modern phones are mainly used in infrastructure mode to provide Internet connectivity. A comparison of the current consumption of different wireless protocols is given in Table 10.1. We argue that Bluetooth offers the most flexibility to connect external sensors due to its availability across different operating systems and manufacturers, even though it is not the most energy-efficient option.

10.2 The Sundroid System

In the following, we present *Sundroid*, which is a personalized sensing system for solar radiation. It measures ultraviolet radiation using a wearable sensor unit in combination with a mobile phone.

10.2.1 Motivation

Sunlight is a driving force for life and, amongst others due to its role in the production of vitamin D, indispensable for our health. However, excessive sunlight exposure can cause sunburn. Moreover, long-term damage, such as premature skin aging and skin cancer, are severe implications that can result from an overexposure to sunlight, even if no immediate signs of sunburn are visible [12]. Both, sunburn as well as the mentioned long-term damages are caused by the ultraviolet (UV) radiation contained in the sunlight. In fact, most of the non-melanoma skin cancers are associated with exposure to UV radiation [110]. A considerable fraction of people is aware of the risks originating from the sun. Nevertheless, the number of skin cancer cases in the United States has constantly increased over the past years [117]. Even though most people know about appropriate protective measures, a large fraction of the population gets sunburned from time to time. Following the result from our survey amongst 785 students (52% male, 48% female) presented in [39], we conclude that inattention and the inability to correctly assess the UV intensity are the two major reasons for excessive UV exposure and, therefore,

for people to get sunburned. To tackle this problem, we propose a wearable system that is able to warn its user once a critical dose of sunlight has been absorbed, and can help to improve people's awareness of the incident radiation in different situations. Furthermore, prospective application scenarios of our system may also include behavioral research, medical therapy, and participatory sensing.

10.2.2 Related Work

A variety of wearable UV dosimeters based on different technologies have been proposed in the past. A rough categorization distinguishes between chemical, biological, and electronic methods to measure UV radiation [58]. As opposed to biological and chemical dosimeters that exhibit an integral characteristic, electronic systems based on photodiodes can measure the UV intensity at a given point in time, e.g. [56]. Thus, they do not only provide higher flexibility with respect to post-processing, but are also well suited to creating plots of the UV intensity over time. Like our sensor system, these devices are designed to be worn by humans. However, all of them measure UV intensity using a single sensor and thus fail to take advantage of the flexibility given by post-processing of individual UVA and UVB values. Moreover, they are primarily designed for offline analysis, and thus do neither provide a direct feedback to the user, nor do they support real-time monitoring. As a consequence, intensity data can only be retrieved after a measurement series has been completed, which is in contrast to our system that constantly transmits the sensor values to a mobile phone, and thus can directly take influence on the user's behavior.

10.2.3 System Design

In this section, we describe the design and implementation of Sundroid. The current prototype system targets a broad range of application scenarios for solar radiation awareness. Due to the modular design, other usage scenarios can easily be addressed by a small modification of the phone application. Sundroid consists of two separate components, as shown in Figure 10.1. A dedicated sensor unit is used to measure UV radiation, while the mobile phone serves as a processing, storage, and communication platform. To provide data relevant to its user, the sensor unit needs to be body wearable. Moreover, to find widespread acceptance, it is inevitable to make the corresponding device as unobtrusive as possible and available at low cost.

Due to application specific action spectra, we decided to independently measure UVA (315–400 nm) and UVB (280–315 nm) intensities. This design decision provides maximal flexibility for the approximation of different spectra. An example is the *Erythema action spectrum*, which describes the

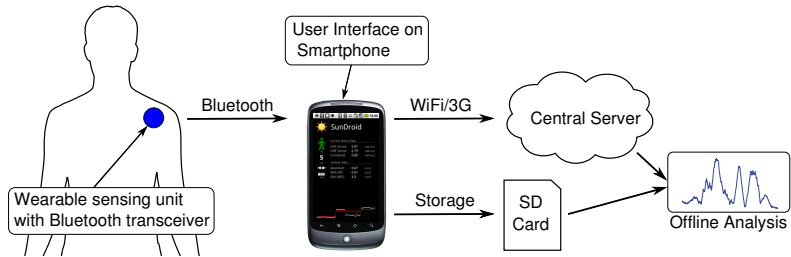


Figure 10.1: Schematic overview of the Sundroid system: A wearable sensing unit containing UV sensors is attached to a user’s clothes or accessories. Sensor data are transferred using Bluetooth to a mobile phone, which serves as a graphical user interface. Optionally, measurement data can be transferred to a central server or stored on a SD card for offline analysis by researchers or medical personnel.

wavelength dependent sensitivity of skin relevant to sunburn. Other examples include the action spectra for DNA damage [132] or skin cancer [28].

To further improve the flexibility with respect to possible applications, we rely on the user’s phone as a generic information hub. On the one hand, the phone can provide further context, such as location and activity patterns. On the other hand, it is used for data processing, presentation and communication. Data exchange between the sensor unit and the phone is implemented over a Bluetooth wireless link in real-time. An important requirement is that the system continuously monitors the solar radiation to which the user is exposed, even when the user is temporarily not in the proximity of the phone. Dependent on the application, the phone can either log the data and present the result to the user, or forward it to a central infrastructure, e.g., for analysis by experts such as researchers or medical personnel.

10.2.4 Wearable Sensing Unit

The wearable sensing unit consists of a custom made printed circuit board and a plastic enclosure, as shown in Figure 10.2. It features a TI MSP430 low-power microcontroller with 4 KBytes program flash and 512 Bytes of RAM, which is connected to a Roving Networks RN-42 low-power Bluetooth module. All components are standard parts available at low prices (tens of dollars). The sensing unit is supplied with power by a 3.7 V lithium-polymer rechargeable battery.

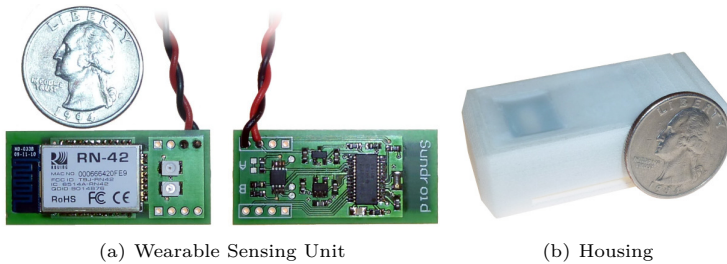


Figure 10.2: The wearable sensing unit consists of a double-sided printed circuit board (left). The Bluetooth module and the UV photodiodes are mounted on the top layer, while the microcontroller and the analog circuits are on the bottom layer. The sensor units are enclosed within a plastic housing (right).

UV Sensors

We equipped the wearable sensing unit with a UVA photodiode (Genicom GUYA-S10GD) and a UVB photodiode (Genicom GUVB-S10GD). Each of the two photodiodes gives an estimation of the current sunlight intensity in their intrinsic spectrum. Depending on the designated use case, we are interested in different spectra. Some medical applications, for example, require the measurement of UVA only. In the context of sunburn, we are interested in the erythema action spectrum, which can be approximated using a weighted linear combination of 9% UVA estimation and 91% UVB estimation with these specific types of UV sensors.

Sensor Calibration

To ensure high accuracy of UV measurements, we have calibrated the sensor units against the erythema-specific reference signal of the World Radiation Center (WRC) in Davos, Switzerland. For the measurements, our sensor devices were horizontally mounted on the roof of the WRC building, next to the reference probes. Figure 10.3 (left) compares the erythema specific responses measured by the high-precision reference equipment with the sensor data collected by our sensor device. The measurement was conducted during a sunny day in winter. Measurement data are reported in the *UV Index* scale.¹ The good match of our sensor data with the reference curve after calibration shows that our sensor unit is able to accurately measure the UV radiation relevant to erythema. In fact, the resulting relative error

¹The UV index is a forecast of the strength of the UV radiation at a particular place, measured on a linear scale between 1 (low) and 11 (very high).

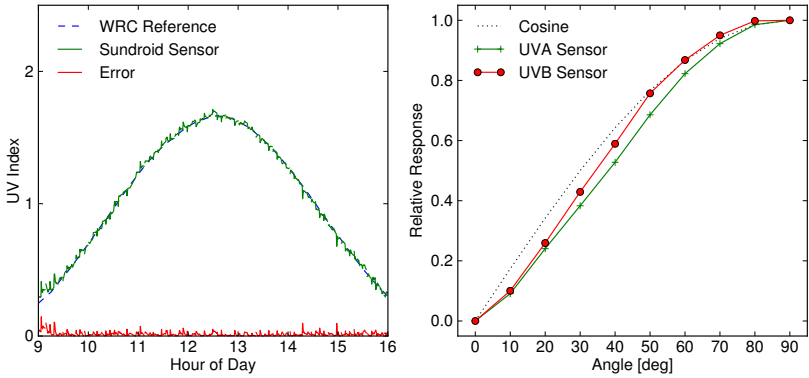


Figure 10.3: Sensor calibration against the reference measurement for the erythema action spectrum at the World Radiation Center (WRC) on a sunny day in winter (left). Relative response of the UVA/B photodiodes placed under a PTFE diffuser when exposed to sunlight at different incident angles (right). It is assumed that at 0° (perpendicular to the incident sunlight) there is only diffuse radiation. The corresponding intensity value has been subtracted from the entire measurement series to account for directed light only.

after calibration is less than 5%. The linear correction factor used for calibration needs to be known by the Sundroid application to convert the raw UV measurements into the human readable UV index scale.

Clearly, the sensor sensitivity is dependent on the direction of the incident radiation. Therefore, we integrate a diffuser made from Polytetrafluoroethylene (PTFE) into the enclosing. Measurement results for different incident angles of sunlight indicate that the dependency on the angle closely follows a cosine curve, as shown in Figure 10.3 (right).

Data Acquisition and Buffering

Measurement of UV radiation using photodiodes can be accomplished with a simple analog circuit. The energy of the incoming light within the active area of the photodiode is converted into a small current. The magnitude of this current depends on the intensity of the light and the wavelength dependent sensitivity of the photodiode with respect to the incoming light's spectrum. Since the resulting currents are very small, we use a transimpedance amplifier circuit to convert the photocurrent into a corresponding voltage, which is continually sampled using the 10-bit analog-digital converter of the microcontroller. Both sensors can be sampled multiple times per second. Depending

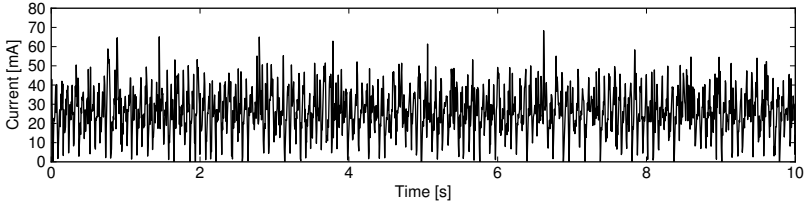
on the availability of a Bluetooth connection to the user's phone, measurement data are either transferred immediately to the phone or buffered locally in the memory of the microcontroller. This helps to avoid data loss when the user temporarily leaves the connection range of the phone. The storage capacity of the microcontroller allows to store several minutes of samples at the highest resolution using a circular buffer. Furthermore, we also accumulate the sum of the sensor readings that have not yet been transferred to the phone. This assures that we can track the total absorbed radiation even if no Bluetooth connection is available for longer periods. However, we might lose some of the individual sensor readings since they have been overwritten in the local buffer with new data.

Power Consumption

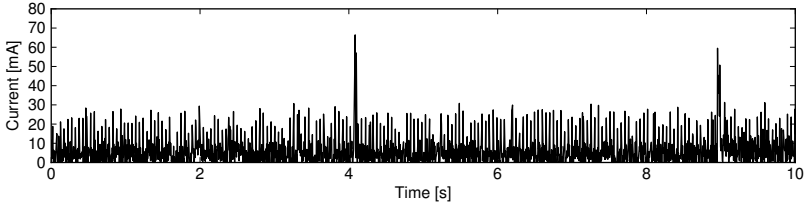
Prolonging battery lifetime is a crucial aspect for wearable applications. Therefore, we carefully designed the sensor unit for low-power operation. The analog sensor circuit and the microcontroller can be put into low-power mode most of the time to reduce the overall power consumption. As a result, the biggest part of the power budget is assigned to the Bluetooth module. By duty-cycling the Bluetooth module when connected (sniff-mode), the total current drawn of the sensor unit can be reduced significantly from 27 to 6 mA, as shown in Figure 10.4. However, the Bluetooth low-power mode has to be supported by both master (phone) and client (sensor). Given the nominal battery capacity of 110 mAh, the sensor unit can be operated up to 18 hours before it has to be recharged. To prevent a sudden power failure, the battery voltage is measured continuously and displayed on the phone. The current hardware design supports operation of the wearable sensing unit during an interval that is in the order of a typical recharge cycle of a modern mobile phone. Hopefully, next generation wireless devices, e.g., transceivers supporting the upcoming Bluetooth 4.0 low energy standard, will allow to tremendously prolong the operation time of the sensor unit using even smaller batteries.

10.2.5 Smartphone Application

Measurement data from the wearable sensor unit are transferred over Bluetooth to a mobile phone running the Google Android operating system. Figure 10.5 depicts the integration of the Sundroid application within the Android operating system. The phone runs a background service, which handles the Bluetooth communication with the sensor device. The wearable sensing unit is polled periodically for new sensor measurements. All sensor readings from the UVA and UVB photo diodes are stored together with the corresponding timestamp and location information provided by the phone. These measurements are made available to the user via the phone in real-time. This



(a) Bluetooth module in standard mode.



(b) Bluetooth module in low-power mode.

Figure 10.4: Current consumption of the Sundroid sensing unit with an active Bluetooth connection for the standard (a) and low-power operation mode (b). The average power consumption measured for the normal operation mode are 27 mA and 6 mA for the low-power mode.

is in contrast to existing UV dosimeters, which only allow for offline analysis. Furthermore, data are written to the phone's SD card for further analysis. If needed, sensor readings can also be sent to a web server, if the phone provides Internet connectivity.

Data Fusion

The advantages of smartphones as a generic processing platform are manifold. On the one hand, they are personal devices that are almost always with their user. On the other hand, they exhibit a variety of readily available sensors, such as 3D-compass, accelerometers, GPS, light sensor, camera, and microphone, from which the user's context can be inferred. In addition, state-of-the-art smartphones feature enormous computing and communication capabilities, which provides a flexible way to process and distribute the gathered sensor data. Currently, we assign the current timestamp and location of the user as provided by the GPS to each sensor reading, which provides valuable context information for the analysis of the measurement data. Beside processing the data of the external sensors, the use of a modern mobile phone allows to gather more context information. For example,

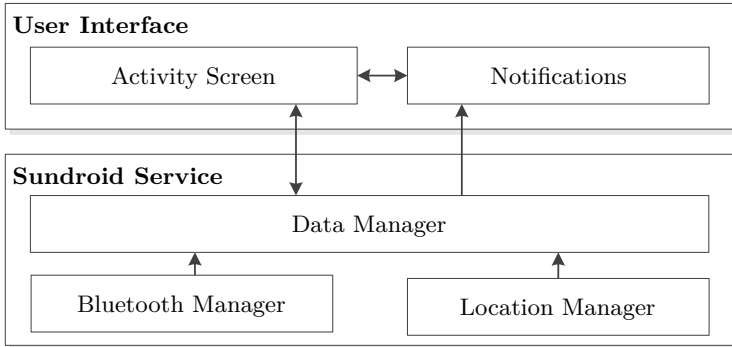


Figure 10.5: Model of data flow within the Sundroid application: Data collection and processing is handled by a background service in Android. The user interacts with the activity screen, which visualizes measurement data and provides adjustment of user specific settings such as skin type. Notifications are triggered when a user interaction is required.

Sundroid fetches the UV Index forecast of the day from the Internet. The predicted UV-index can be used to inform the user on time about necessary precautions, such as to take along appropriate sunscreen or clothing.

Data Processing and Visualization

A considerable difference between stand-alone sensors and mobile phone based sensor systems is that the data processing and displaying can be easily adapted to the needs of the use case on phone based systems. In the case of Sundroid, this allows us to adapt the system to various application scenarios. The Sundroid application features two different user interfaces tailored to the specific use case, as depicted in Figure 10.6. The *Simple View* provides basic information about the user's exposure to sunlight at a glance. In contrast, the *Advanced View* gives detailed information about current sensor readings and total accumulated energy. This view is mainly intended for use by technical experts (e.g. physicians or researchers) for evaluation purposes. Additional views, which are tailored to a more specific use case, can be added easily thanks to the extensibility of the Android platform.

Pilot Study

In order to demonstrate the practical usability of our system and to investigate the effects of sensor placement in a real-world setting, we have conducted a one-day measurement of solar exposure during winter sports activity, namely snowboarding. The test subject was equipped with three

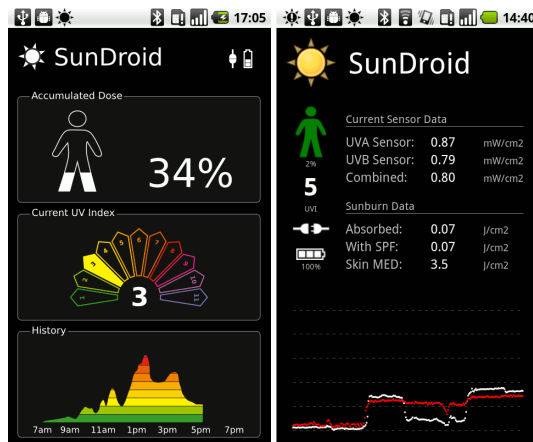


Figure 10.6: Screenshot of the Sundroid activity screens on the phone. The *Simple View* provides basic information about sunlight exposure at a glance (left), while the *Advanced View* provides additional detail about the UVA/UVB measurements (right).

wearable sensors attached to the head, shoulder and chest. Post-processing of measurement data stored on the SD card allows to reconstruct the timeline of solar exposure, as shown in Figure 10.7. The internal GPS receiver of the mobile phone used during the experiment provides valuable context information, for instance the time, position (see Figure 10.8) and altitude of the test subject. This information allows to relate the amount of solar radiation to a place and exact time point, which is beneficial for behavioral research. We observe two time periods during which the test subject was not exposed to solar radiation. The first period at around noon was during the lunch break in an indoor restaurant area, the second period corresponds to a trip in a gondola lift with closed cabins. In addition to the user's location provided by the GPS receiver, sampling the integrated sensors of the mobile phone, e.g., compass, accelerometer or gyroscope, allows to gather valuable context information for the further analysis of a user's exposure to UV radiation.

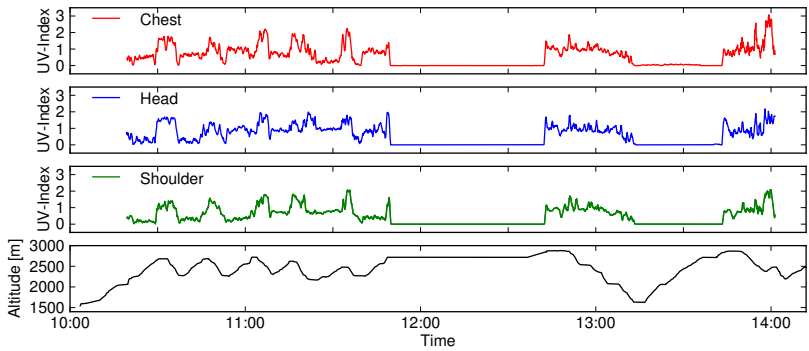


Figure 10.7: Measurement of the UV intensity of three Sundroid sensors placed at the chest, shoulder, and head of a test subject during 4 hours of snowboarding and the corresponding elevation profile provided by the GPS receiver.

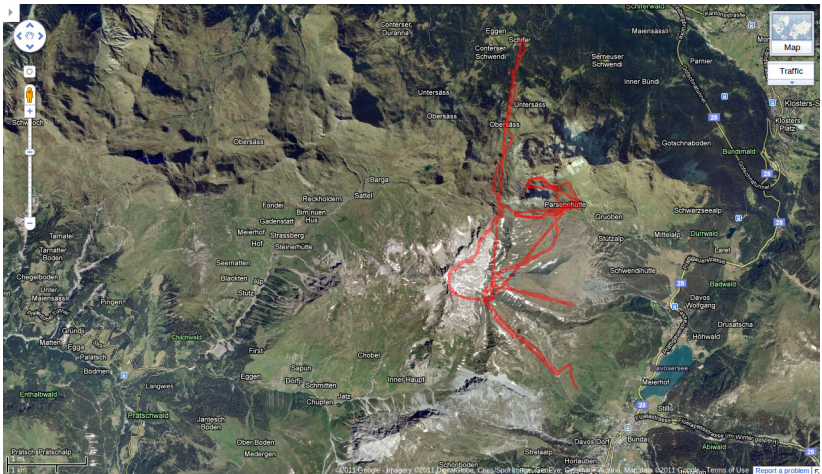


Figure 10.8: Visualization of the test subject's location during the 4 hour measurement in the ski resort of Davos, Switzerland (Screenshot: Google Maps).

11

Conclusions

The application domain of modern wireless embedded system extends beyond environmental monitoring applications, as explored in the early years of sensor network research. In classical data gathering architectures, sensor nodes forward their measurements along a tree to a gateway node, which bridges the gap between the sensor network and the outside world. Since such networks are designed and optimized for a specific purpose, integrating additional sensors or different node platforms requires a significant amount of work for programming and configuration.

In the last few years, sensor platforms have become ever more powerful and the ease of interoperability has increased by utilizing standard communication hardware, e.g., radio transceivers according to the IEEE 802.15.4 standard. To leverage this potential, we proposed an approach to interconnect different sensor and actuator nodes using standard Internet protocols in Chapter 9. This makes it feasible to build a sensor network across various types of devices and underlying communication protocols. Even though we are targeting tiny sensor nodes with limited memory and processing power, we employ web services to interact with sensor nodes. To this end, every node implements a small web server on top of a TCP/IP stack to provide access to sensor data and control of actuators using standard protocols. Thereby, data is exchanged using a simple text-based format, which is human-readable. Other clients, such as a central unit or any other device in such a network, can access the list of resources and services offered by a device using a simple

browsing mechanism. Moreover, device discovery based on multicast DNS messages enables the system to integrate new devices without additional configuration effort.

The main technical challenge of this work is the integration of resource-constrained sensor networks into IP-based networks. We demonstrate the feasibility of our approach by the implementation of a prototype system using TinyOS on a mote-class node platform. However, the integration of ubiquitous wireless embedded systems into public IP-based networks will require sophisticated access control mechanisms to ensure that sensor data are protected from unauthorized access.

Similarly to the availability of more powerful sensor node platforms, mobile phones have become a steady companion that offers much more than just the ability to make a telephone call. The mobile phone is the new “Swiss Army knife” of the 21st century offering technical capabilities for communication, computation, storage, synchronization, and localization. Therefore, it is also an ideal candidate to take the role of a gateway or central hub for people-centric sensing applications.

In Chapter 10, we present a system architecture to integrate wearable wireless sensors with a mobile phone. Exemplarily, we present the *Sundroid* system, which is designed to increase the personal awareness of solar radiation. In contrast to existing UV dosimeters that are designed for offline analysis only, our system tracks the wearer’s sunlight exposure in real-time. The prototype system is designed in a modular fashion and consists of a small sensor unit, which measures the UV radiation, and a smartphone application, which collects, processes, and displays the sensor unit’s measurements. We believe that the mobile phone will become increasingly important as a platform for personalized sensing applications. Yet, there remain issues that need to be address by future work, such as decreasing the size, cost, and power consumption of wearable sensors. The most challenging problem will be to reduce the current drain of the wireless communication interface in order to operate the system for weeks or even several months with a small battery.

Bibliography

- [1] *Fundamentals of Quartz Oscillators*. Application note 200-2, Hewlett-Packard, 1997.
- [2] D. W. Allan, N. Ashby, and C. C. Hodge. The Science of Timekeeping (Hewlett-Packard Application Note 1289), 1997.
- [3] M. Allen, L. Girod, R. Newton, S. Madden, D. Blumstein, and D. Estrin. VoxNet: An Interactive, Rapidly-Deployable Acoustic Monitoring Platform. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 371–382, 2008.
- [4] M. Azizyan and R. R. Choudhury. SurroundSense : Mobile Phone Localization via Ambience Fingerprinting. In *Proceedings of the 15th International Conference on Mobile Computing and Networking (MobiCom)*, pages 261–272, 2009.
- [5] P. Bahl and V. Padmanabhan. RADAR: An In-Building RF-based User Location and Tracking System. In *Proceedings of the 19th International Conference on Computer Communications (INFOCOM)*, pages 775–784, 2000.
- [6] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the Time-Complexity of Broadcast in Multi-hop Radio Networks: An Exponential Gap between Determinism and Randomization. *Journal of Computer and System Sciences*, 45(1):104–126, 1992.
- [7] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The Hitchhiker’s Guide to Successful Wireless Sensor Network Deployments. In *Proceedings of the 6th International Conference on Embedded Network Sensor Systems (SenSys)*, pages 43–56, 2008.
- [8] A. Basu, J. Gao, J. S. B. Mitchell, and G. Sabhnani. Distributed Localization Using Noisy Distance and Angle Information. In *Proceedings of the 7th International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 262–273, 2006.

- [9] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Woehrle, and M. Yuceel. PermaDAQ: A Scientific Instrument for Precision Sensing and Data Recovery in Environmental Extremes. In *Proceedings of the 8th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 265–276, 2009.
- [10] B. Bhatta. *Global Navigation Satellite Systems: Insights into GPS, GLONASS, Galileo, Compass and Others*. CRC Press, 2011.
- [11] S. Biaz and J. Welch. Closed Form Bounds for Clock Synchronization Under Simple Uncertainty Assumptions. *Information Processing Letters*, 80(3):151–157, 2001.
- [12] D. E. Brash, J. A. Rudolph, J. A. Simon, A. Lin, G. J. McKenna, H. P. Baden, A. J. Halperin, and J. Pontén. A role for sunlight in skin cancer: UV-induced p53 mutations in squamous cell carcinoma. *Proceedings of the National Academy of Sciences of the United States of America*, 88(22):10124–8, Nov. 1991.
- [13] J. Bruck, J. Gao, and A. A. Jiang. Localization and Routing in Sensor Networks by Local Angle Information. In *Proceedings of the 6th International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 1–31, 2005.
- [14] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less Low-Cost Outdoor Localization for Very Small Devices. *IEEE Personal Communications*, 7(5):28–34, 2000.
- [15] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. Srivastava. Participatory Sensing. In *Proceedings of the Workshop on World-Sensor-Web (WSW)*, pages 1–5, 2006.
- [16] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: Ultra-Low Power Data Gathering in Sensor Networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 450–459, 2007.
- [17] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. a. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.-S. Ahn. The Rise of People-Centric Sensing. *IEEE Internet Computing*, 12(4):12–21, July 2008.
- [18] M. Cao, A. Morse, and B. Anderson. Reaching a Consensus in a Dynamically Changing Environment: Convergence Rates, Measurement Delays and Asynchronous Events. *SIAM Journal on Control and Optimization*, 47(2):601–623, 2008.

- [19] M. Ceriotti, L. Mottola, G. P. Picco, A. L. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon. Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment. In *Proceedings of the 8th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 277–288, 2009.
- [20] H.-l. Chang, J.-b. Tian, T.-T. Lai, H.-H. Chu, and P. Huang. Spinning Beacons for Precise Indoor Localization. In *Proceedings of the 6th International Conference on Embedded Networked Sensor Systems (SenSys)*, page 127, 2008.
- [21] Y. Chen, Q. Wang, M. Chang, and A. Terzis. Ultra-Low Power Time Synchronization Using Passive Radio Receivers. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 235–245, 2011.
- [22] I. Chlamtac. The Wave Expansion Approach to Broadcasting in Multihop Radio Networks. *IEEE Transactions on Communications*, 39(3):426–433, Mar. 1991.
- [23] I. Chlamtac and S. Kutten. On Broadcasting in Radio Networks—Problem Analysis and Protocol Design. *IEEE Transactions on Communications*, 33(12):1240–1246, 1985.
- [24] Communication from the European Commission. Energy Efficiency: Delivering the 20% target. (772), Nov. 2008.
- [25] D. E. Culler and G. Tolle. Embedded Web Services: Making Sense out of Diverse Sensors. *Sensors*, May 2007.
- [26] A. Czumaj and W. Rytter. Broadcasting Algorithms in Radio Networks with Unknown Topology. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 492–501, 2003.
- [27] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler. sMAP: A Simple Measurement and Actuation Profile for Physical Information. In *Proceedings of the 8th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 197–210, 2010.
- [28] F. R. de Gruijl, H. J. C. M. Sterenborg, P. D. Forbes, R. E. Davies, C. Cole, G. Kelfkens, H. van Weelden, H. Slaper, and J. C. van der Leun. Wavelength Dependence of Skin Cancer Induction by Ultraviolet Irradiation of Albino Hairless Mice. *Cancer Research*, 53(1):53–60, 1993.

- [29] H. Dubois-Ferriere, R. Meier, L. Fabre, and P. Metrailler. TinyNode: A Comprehensive Platform for Wireless Sensor Network Applications. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 358–365, 2006.
- [30] A. Dunkels. Full TCP/IP for 8 Bit Architectures. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 85–98, 2003.
- [31] A. Dunkels and J. P. Vasseur. IP for Smart Objects. Ipson alliance white paper #1, Sept. 2008.
- [32] M. Dyer, J. Beutel, T. Kalt, P. Oehen, L. Thiele, K. Martin, and P. Blum. Deployment Support Network - A Toolkit for the Development of WSNs. In *Proceedings of the 4th European Conference on Wireless Sensor Networks (EWSN)*, pages 195–211, 2007.
- [33] N. Eagle and A. (Sandy) Pentland. Reality Mining: Sensing Complex Social Systems. *Personal and Ubiquitous Computing*, 10(4):255–268, 2006.
- [34] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. The BikeNet Mobile Sensing System for Cyclist Experience Mapping. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 87–101, 2007.
- [35] J. Elson, L. Girod, and D. Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 147–163, 2002.
- [36] J. Elson and K. Römer. Wireless Sensor Networks: A New Regime for Time Synchronization. *ACM SIGCOMM Computer Communication Review*, 33(1):149–154, 2003.
- [37] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan. The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 29–39, 2008.
- [38] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the Physical World with Pervasive Networks. *IEEE Pervasive Computing*, 1(1):59–69, Jan. 2002.

- [39] T. Fahrni, M. Kuhn, P. Sommer, R. Wattenhofer, and S. Welten. Sundroid: Solar Radiation Awareness with Smartphones. In *Proceedings of the 13th International Conference on Ubiquitous Computing (Ubi-comp)*, pages 365–374, 2011.
- [40] R. Fan and N. Lynch. Gradient Clock Synchronization. In *Proceedings of the 23th Symposium on Principles of Distributed Computing (PODC)*, pages 320–327, 2004.
- [41] J. Farrell and M. Barth. *The Global Positioning System & Inertial Navigation*. McGraw-Hill, 1998.
- [42] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient Network Flooding and Time Synchronization with Glossy. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 73–84, 2011.
- [43] R. T. Fielding and R. N. Taylor. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, 2002.
- [44] R. Flury. *Routing on the Geometry of Wireless Ad Hoc Networks*. Ph.d. thesis, ETH Zurich, 2009.
- [45] R. Flury and R. Wattenhofer. Slotted Programming for Sensor Networks. In *Proceedings of the 9th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 24–34, 2010.
- [46] S. Ganeriwal, R. Kumar, and M. Srivastava. Timing-sync Protocol for Sensor Networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 138–149, 2003.
- [47] L. Girod, M. Lukac, V. Trifa, and D. Estrin. The Design and Implementation of a Self-Calibrating Distributed Acoustic Sensing Platform. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 71–84, 2006.
- [48] L. Gu, T. Abdelzaher, B. H. Krogh, D. Jia, P. Vicaire, T. Yan, L. Luo, A. Tirumala, Q. Cao, T. He, and J. A. Stankovic. Lightweight Detection and Classification for Wireless Sensor Networks in Realistic Environments. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 205–217, 2005.
- [49] D. Guinard and V. Trifa. Towards the Web of Things: Web Mashups for Embedded Devices. In *Proceedings of Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM)*, 2009.

- [50] D. Guinard, V. Trifa, and E. Wilde. A Resource Oriented Architecture for the Web of Things. In *Proceedings of the 2nd International Conference on the Internet of Internet of Things (IOT)*, pages 1–8, 2010.
- [51] I. Guttman. *Introductory Engineering Statistics*. John Wiley & Sons, 1982.
- [52] D. Hahnel, W. Burgard, D. Fox, K. Fishkin, and M. Philipose. Mapping and Localization with RFID Technology. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 1015–1020, 2004.
- [53] V. Handziski, A. Köpke, A. Willig, and A. Wolisz. TWIST: A Scalable and Reconfigurable Testbed for Wireless Indoor Experiments with Sensor Networks. In *Proceedings of the 2nd International Workshop on Multi-Hop Ad Hoc Networks: From Theory to Reality (REALMAN)*, pages 63–70, 2006.
- [54] A. Harter, A. Hopper, P. Steggle, A. Ward, and P. Webster. The Anatomy of a Context-Aware Application. In *Proceedings of the 5th International Conference on Mobile Computing and Networking (MobiCom)*, pages 59–68, 1999.
- [55] J. Helander. Deeply Embedded XML Communication: Towards an Interoperable and Seamless World. In *Proceedings of the 5th International Conference on Embedded Software (EMSOFT)*, pages 62–67, 2005.
- [56] J. Heydenreich and H. C. Wulf. Miniature Personal Electronic UVR Dosimeter with Erythema Response and Time-stamped Readings in a Wristwatch. *Photochemistry and photobiology*, 81(5):1138–1144, 2005.
- [57] J. Hill and D. Culler. Mica: A Wireless Platform for Deeply Embedded Networks. *IEEE Micro*, 22(6):12–24, 2002.
- [58] G. Horneck. Quantification of the biological effectiveness of environmental UV radiation. *Journal of Photochemistry and Photobiology B: Biology*, 31(1-2):43–49, Nov. 1995.
- [59] J. Hui and D. Culler. The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 81–94, 2004.

- [60] J. Hui, D. Culler, and S. Chakrabarti. 6LoWPAN: Incorporating IEEE 802.15.4 into the IP Architecture. Ipson alliance white paper #3, Jan. 2009.
- [61] J. W. Hui and D. E. Culler. IP is Dead, Long Live IP for Wireless Sensor Networks. In *Proceedings of 6th Conference on Embedded Network Sensor Systems (SenSys)*, pages 15–28, 2008.
- [62] X. Jiang, S. Dawson-Haggerty, P. Dutta, and D. Culler. Design and Implementation of a High-Fidelity AC Metering Network. In *Proceedings of 8th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 253–264, 2009.
- [63] T. Jones. *Splitting the Second : The Story of Atomic Time*. Taylor & Francis, 2000.
- [64] R. Jurdak, P. Corke, D. Dharman, and G. Salagnac. Adaptive GPS Duty Cycling and Radio Ranging for Energy-efficient Localization. In *Proceedings of the 8th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 57–70, 2010.
- [65] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health Monitoring of Civil Infrastructures using Wireless Sensor Networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 254–263, 2007.
- [66] H. Kopetz and W. Ochsenreiter. Clock Synchronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, 100(8):933–940, 1987.
- [67] E. Kranakis, H. Singh, and J. Urrutia. Compass Routing on Geometric Networks. In *Proceedings of the 11th Canadian Conference on Computational Geometry (CCCg)*, pages 51–54, 1999.
- [68] F. Kuhn and R. Oshman. Gradient Clock Synchronization Using Reference Broadcasts. *Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS)*, 5923:204–218, 2009.
- [69] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric Ad-Hoc Routing: Of Theory and Practice. In *Proceedings of the 22th Symposium on Principles of Distributed Computing (PODC)*, pages 63–72, 2003.
- [70] Y.-S. Kuo, S. Verma, T. Schmid, and P. Dutta. Hijacking Power and Bandwidth from the Mobile Phone’s Audio Interface. In *Proceedings of the 1st Symposium on Computing for Development (DEV)*, 2010.

- [71] B. Kusý. *Spatiotemporal Coordination in Wireless Sensor Networks*. Ph.d. thesis, Vanderbilt University, 2007.
- [72] B. Kusý, P. Dutta, P. Levis, M. Maróti, A. Lédeczi, and D. Culler. Elapsed Time on Arrival: A Simple and Versatile Primitive for Canonical Time Synchronization Services. *Int. J. Ad Hoc Ubiquitous Comput.*, 1(4):239–251, 2006.
- [73] B. Kusý, A. Lédeczi, and X. Koutsoukos. Tracking Mobile Nodes Using RF Doppler Shifts. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 29–42, 2007.
- [74] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978.
- [75] D. S. Landes. *Revolution in Time: Clocks and the Making of the Modern World*. Belknap Press, 2000.
- [76] N. D. Lane, S. B. Eisenman, M. Musolesi, E. Miluzzo, and A. T. Campbell. Urban Sensing Systems: Opportunistic or Participatory? In *Proceedings of the 9th Workshop on Mobile Computing Systems and Applications (HotMobile)*, pages 11–16, 2008.
- [77] K. Langendoen, A. Baggio, and O. Visser. Murphy Loves Potatoes: Experiences from a Pilot Sensor Network Deployment in Precision Agriculture. In *Proceedings of the 20th International Parallel & Distributed Processing Symposium (IPDPS)*, pages 1–8, 2006.
- [78] C. Lenzen. *Synchronization and Symmetry Breaking in Distributed Systems*. Ph.d. thesis, ETH Zurich, 2011.
- [79] C. Lenzen, T. Locher, and R. Wattenhofer. Tight Bounds for Clock Synchronization. *Journal of the ACM*, 57(2):1–42, 2010.
- [80] C. Lenzen, P. Sommer, and R. Wattenhofer. Optimal Clock Synchronization in Networks. In *Proceedings of the 7th Conference on Embedded Networked Sensor Systems (SenSys)*, pages 225–238, 2009.
- [81] P. Levis and D. Gay. *TinyOS Programming*. Cambridge University Press, 2009.
- [82] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*, pages 15–28, 2004.

- [83] L. Li, G. Xing, L. Sun, W. Huangfu, R. Zhou, and H. Zhu. Exploiting FM Radio Data System for Adaptive Clock Calibration in Sensor Networks. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 169–182, 2011.
- [84] Q. Li and D. Rus. Global Clock Synchronization in Sensor Networks. In *Proceedings of the 23rd International Conference on Computer Communications (INFOCOM)*, pages 564–574, 2004.
- [85] S. Lin, J. Zhang, G. Zhou, L. Gu, T. He, and J. A. Stankovic. ATPC : Adaptive Transmission Power Control for Wireless Sensor Networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 223–236, 2006.
- [86] K. Lippincott, U. Eco, and E. H. Gombrich. *The Story of Time*. Merrell Publishers, 2003.
- [87] T. Locher and R. Wattenhofer. Oblivious Gradient Clock Synchronization. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC)*, pages 520–533, 2006.
- [88] K. Lorincz, B.-r. Chen, G. W. Challen, A. R. Chowdhury, S. Patel, P. Bonato, and M. Welsh. Mercury: A Wearable Sensor Network Platform for High-Fidelity Motion Analysis. In *Proceedings of the 7th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 183–196, 2009.
- [89] J. Lu and K. Whitehouse. Flash Flooding: Exploiting the Capture Effect for Rapid Flooding in Wireless Sensor Networks. In *Proceedings of the 28th Conference on Computer Communications (INFOCOM)*, pages 2491–2499, 2009.
- [90] J. Lundelius and N. A. Lynch. An Upper and Lower Bound for Clock Synchronization. *Information and Control*, 62(2/3):190–204, 1984.
- [91] M. Maróti, B. Kusý, G. Simon, and A. Lédeczi. The Flooding Time Synchronization Protocol. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 39–49, 2004.
- [92] M. Maróti, P. Völgyesi, S. Dóra, B. Kusý, A. Nádas, A. Lédeczi, G. Balogh, and K. Molnár. Radio Interferometric Geolocation. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 1–12, 2005.
- [93] P. Marwedel. *Embedded System Design*. Springer, 2003.

- [94] L. Meier and L. Thiele. Brief Announcement: Gradient Clock Synchronization in Sensor Networks. In *Proceedings of the 24th Symposium on Principles of Distributed Computing (PODC)*, pages 238–238, 2005.
- [95] D. L. Mills. Internet Time Synchronization: the Network Time Protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, 1991.
- [96] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell. Sensing meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application. In *Proceedings of the 6th International Conference on Embedded Network Sensor Systems (SenSys)*, pages 337–350, 2008.
- [97] P. Mohan, V. Padmanabhan, and R. Ramjee. Nericell: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones. In *Proceedings of the 6th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 323–336, 2008.
- [98] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), Sept. 2007.
- [99] D. Moore, J. Leonard, D. Rus, and S. Teller. Robust Distributed Network Localization with Noisy Range Measurements. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 50–61, 2004.
- [100] T. Moscibroda, R. O’Dell, M. Wattenhofer, and R. Wattenhofer. Virtual Coordinates for Ad hoc and Sensor Networks. In *Proceedings of the Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pages 8–16, 2004.
- [101] T. Moscibroda, P. von Rickenbach, and R. Wattenhofer. Analyzing the Energy-Latency Trade-Off During the Deployment of Sensor Networks. In *Proceedings of the 25th International Conference on Computer Communications (INFOCOM)*, pages 1–13, 2006.
- [102] D. Niculescu and B. Nath. Ad Hoc Positioning System (APS) Using AOA. In *Proceedings of the 22nd International Conference on Computer Communications (INFOCOM)*, pages 1734–1743, 2003.
- [103] D. Niculescu and B. Nath. DV Based Positioning in Ad Hoc Networks. *Telecommunication Systems*, 22(1):267–280, 2003.

- [104] G. Oberholzer, P. Sommer, and R. Wattenhofer. SpiderBat: Augmenting Wireless Sensor Networks with Distance and Angle Information. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 211–222, 2011.
- [105] M. O’Dell, R. O’Dell, M. Wattenhofer, and R. Wattenhofer. Lost in Space Or Positioning in Sensor Networks. In *Proceedings of the Workshop on Real-World Wireless Sensor Networks (REALWSN)*, 2005.
- [106] R. Ostrovsky and B. Patt-Shamir. Optimal and Efficient Clock Synchronization under Drifting Clocks. In *Proceedings of the 18th Symposium on Principles of Distributed Computing (PODC)*, pages 3–12, 1999.
- [107] S. Patel, K. Lorincz, R. Hughes, N. Huggins, J. Growdon, D. Standaert, M. Akay, J. Dy, M. Welsh, and P. Bonato. Monitoring Motor Fluctuations in Patients with Parkinson’s Disease using Wearable Sensors. *IEEE Transactions on Information Technology in Biomedicine*, 13(6):864–73, Nov. 2009.
- [108] C. Pautasso, O. Zimmermann, and F. Leymann. RESTful Web Services vs. “Big” Web Services: Making the Right Architectural Decision. In *Proceedings of the 17th International Conference on World Wide Web (WWW)*, pages 805–814, 2008.
- [109] S. V. Pemmaraju and I. A. Pirwani. Good Quality Virtual Realization of Unit Ball Graphs. In *Proceedings of the 15th European Conference on Algorithms (ESA)*, pages 311–322, 2007.
- [110] E. D. Pleasance, R. K. Cheetham, P. J. Stephens, D. J. McBride, S. J. Humphray, C. D. Greenman, I. Varela, M.-L. Lin, G. R. Ordóñez, G. R. Bignell, K. Ye, J. Alipaz, M. J. Bauer, D. Beare, A. Butler, R. J. Carter, L. Chen, A. J. Cox, S. Edkins, P. I. Kokko-Gonzales, N. a. Gormley, R. J. Grocock, C. D. Haudenschild, M. M. Hims, T. James, M. Jia, Z. Kingsbury, C. Leroy, J. Marshall, A. Menzies, L. J. Mudie, Z. Ning, T. Royce, O. B. Schulz-Trieglaff, A. Spiridou, L. a. Stebbings, L. Szajkowski, J. Teague, D. Williamson, L. Chin, M. T. Ross, P. J. Campbell, D. R. Bentley, P. A. Futreal, and M. R. Stratton. A comprehensive catalogue of somatic mutations from a human cancer genome. *Nature*, 463(7278):191–6, Jan. 2010.
- [111] J. Polastre, J. Hill, and D. Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 95–107, 2004.

- [112] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *Proceedings 4th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 364–369, 2005.
- [113] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *Proceedings of the 6th International Conference on Mobile Computing and Networking (MobiCom)*, pages 32–43, 2000.
- [114] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. Tiny Web Services: Design and Implementation of Interoperable and Evolvable Sensor Networks. In *Proceedings of 6th Conference on Embedded Network Sensor Systems (SenSys)*, pages 253–266, 2008.
- [115] N. B. Priyantha, A. K. Miu, H. Balakrishnan, and S. Teller. The Cricket Compass for Context-Aware Mobile Applications. In *Proceedings of the 7th International Conference on Mobile Computing and Networking (MobiCom)*, pages 1–14, 2001.
- [116] R. K. Rana, C. T. Chou, S. S. Kanhere, N. Bulusu, and W. Hu. Ear-Phone: An End-to-End Participatory Urban Noise Mapping System. In *Proceedings of the 9th International Conference on Information Processing in Sensor Networks (IPSN)*, 2010.
- [117] H. W. Rogers, M. A. Weinstock, A. R. Harris, M. R. Hinckley, S. R. Feldman, A. B. Fleischer, and B. M. Coldiron. Incidence Estimate of Nonmelanoma Skin Cancer in the United States, 2006. *Archives of Dermatology*, 146(3):283–7, Mar. 2010.
- [118] K. Römer. Time Synchronization in Ad Hoc Networks. In *Proceedings of the 2nd International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 173–182, 2001.
- [119] K. Römer. The Lighthouse Location System for Smart Dust. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services (MobiSys)*, number 5005, pages 15–30, 2003.
- [120] A. Rowe, V. Gupta, and R. R. Rajkumar. Low-power Clock Synchronization using Electromagnetic Energy Radiating from AC Power Lines. In *Proceedings of the 7th Conference on Embedded Networked Sensor Systems (SenSys)*, pages 211–224, 2009.
- [121] J. Sallai, B. Kusý, A. Lédeczi, and P. Dutta. On the Scalability of Routing Integrated Time Synchronization. *Proceedings of the 3rd European Workshop on Wireless Sensor Networks (EWSN)*, pages 115–131, 2006.

- [122] C. Savarese, J. M. Rabaey, and K. Langendoen. Robust Positioning Algorithms for Distributed Ad-Hoc Wireless Sensor Networks. In *Proceedings of the USENIX Annual Technical Conference (ATEC)*, pages 317–327, 2002.
- [123] A. Savvides, C. Han, and M. Srivastava. Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors. In *Proceedings of the 7th International Conference on Mobile Computing and Networking (MobiCom)*, pages 166–179, 2001.
- [124] A. Savvides, H. Park, and M. B. Srivastava. The Bits and Flops of the N-hop Multilateration Primitive For Node Localization Problems. In *Proceedings of the 1st International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 112–121, 2002.
- [125] L. Schenato and G. Gamba. A Distributed Consensus Protocol for Clock Synchronization in Wireless Sensor Network. In *Proceedings of the 46th Conference on Decision and Control (CDC)*, pages 2289–2294, 2007.
- [126] T. Schmid. *Time in Wireless Embedded Systems*. Ph.d. thesis, UC Los Angeles, 2009.
- [127] T. Schmid, Z. Charbiwala, Z. Anagnostopoulou, M. Srivastava, and P. Dutta. A Case Against Routing-Integrated Time Synchronization. In *Proceedings of the 8th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 267–280, 2010.
- [128] T. Schmid, Z. Charbiwala, R. Shea, and M. Srivastava. Temperature Compensated Time Synchronization. *IEEE Embedded Systems Letters*, 1(2):37–41, 2009.
- [129] T. Schmid, P. Dutta, and M. B. Srivastava. High-Resolution, Low-Power Time Synchronization an Oxymoron No More. In *Proceedings of the 9th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 151–161, 2010.
- [130] J. Schneider and R. Wattenhofer. A Log-Star Distributed Maximal Independent Set Algorithm for Growth-Bounded Graphs. In *Proceedings of the 27th Symposium on Principles of Distributed Computing (PODC)*, pages 35–44, 2008.
- [131] L. Schor, P. Sommer, and R. Wattenhofer. Towards a Zero-Configuration Wireless Sensor Network Architecture for Smart Buildings. In *Proceedings of the 1st Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)*, pages 31–36, 2009.

- [132] R. B. Setlow. The Wavelengths in Sunlight Effective in Producing Skin Cancer: A Theoretical Analysis. *Proceedings of the National Academy of Sciences of the United States of America*, 71(9):3363–6, Sept. 1974.
- [133] Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz. Localization from Mere Connectivity. In *Proceedings of the 4th International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 201–212, 2003.
- [134] K. Shilton, N. Ramanathan, S. Reddy, V. Samanta, J. Burke, D. Estrin, M. Hansen, and M. Srivastava. Participatory Design of Sensing Networks : Strengths and Challenges. In *Proceedings of the 10th Conference on Participatory Design (PDC)*, pages 282–285, 2008.
- [135] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusý, A. Nádas, G. Pap, J. Sallai, and K. Frampton. Sensor Network-Based Countersniper System. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 1–12, 2004.
- [136] R. Solis, V. S. Borkar, and P. R. Kumar. A New Distributed Time Synchronization Protocol for Multihop Wireless Networks. In *Proceedings of the 45th Conference on Decision and Control (CDC)*, pages 2734–2739, 2006.
- [137] P. Sommer and R. Wattenhofer. Symmetric Clock Synchronization in Sensor Networks. In *Proceedings of the Workshop on Real-World Wireless Sensor Networks (REALWSN)*, pages 11–15, 2008.
- [138] P. Sommer and R. Wattenhofer. Gradient Clock Synchronization in Wireless Sensor Networks. In *8th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 37–48, 2009.
- [139] T. K. Srikanth and S. Toueg. Optimal Clock Synchronization. *Journal of the ACM*, 34(3):626–645, 1987.
- [140] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. Understanding the Causes of Packet Delivery Success and Failure in Dense Wireless Sensor Networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 419–420, 2006.
- [141] F. Stann, J. Heidemann, R. Shroff, and M. Zaki. RBP : Robust Broadcast Propagation in Wireless Networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 85–98, 2006.

- [142] R. Stoleru, T. He, J. A. Stankovic, and D. Luebke. A High-Accuracy, Low-Cost Localization System for Wireless Sensor Networks. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 13–26, 2005.
- [143] R. Stoleru, P. Vicaire, T. He, and J. A. Stankovic. StarDust: A Flexible Architecture for Passive Localization in Wireless Sensor Networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 57–70, 2006.
- [144] A. Syed and J. Heidemann. Time Synchronization for High Latency Acoustic Networks. In *Proceedings of the 25th International Conference on Computer Communications (INFOCOM)*, pages 1–12, 2006.
- [145] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An Analysis of a Large Scale Habitat Monitoring Application. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 214–226, 2004.
- [146] G. Tolle, D. Gay, W. Hong, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, and P. Buonadonna. A Macroscope in the Redwoods. In *Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys)*, pages 51–63, 2005.
- [147] J. van Greunen and J. Rabaey. Lightweight Time Synchronization for Sensor Networks. In *Proceedings of the 2nd International Conference on Wireless Sensor Networks and Applications (WSNA)*, pages 11–19, 2003.
- [148] P. Vicaire, T. F. Abdelzaher, T. He, Q. Cao, T. Yan, G. Zhou, L. Gu, L. Luo, R. Stoleru, and J. a. Stankovic. Achieving Long-Term Surveillance in VigilNet. *ACM Transactions on Sensor Networks*, 5(1):1–39, Feb. 2009.
- [149] M. Welsh. Sensor Networks for the Sciences. *Communications of the ACM*, 53(11):36–39, Nov. 2010.
- [150] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and Yield in a Volcano Monitoring Sensor Network. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 381–396, 2006.
- [151] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: A Wireless Sensor Network Testbed. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 483–488, 2005.

- [152] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 142–153, 2005.
- [153] K. Whitehouse and D. Culler. A Robustness Analysis of Multi-hop Ranging-based Localization Approximations. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 317–325, 2006.
- [154] K. Whitehouse, F. Jiang, A. Woo, C. Karlof, and D. Culler. Sensor Field Localization: A Deployment and Empirical Analysis. 2004.
- [155] K. Whitehouse, C. Karlof, and D. Culler. A Practical Evaluation of Radio Signal Strength for Ranging-based Localization. *SIGMOBILE Mob. Comput. Commun. Rev.*, 11(1):41–52, 2007.
- [156] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the Capture Effect for Collision Detection and Recovery. In *Proceedings of the 2nd Workshop on Embedded Networked Sensors (EmNetS)*, pages 45–52, 2005.
- [157] J. Wolfowitz. Products of Indecomposable, Aperiodic, Stochastic Matrices. *Proceedings of the American Mathematical Society*, 14(5):733–737, 1963.
- [158] A. Wood, J. Stankovic, G. Virone, L. Selavo, Z. He, Q. Cao, T. Doan, Y. Wu, L. Fang, and R. Stoleru. Context-Aware Wireless Sensor Networks for Assisted Living and Residential Monitoring. *IEEE Network*, 22(4):26–33, July 2008.
- [159] D. Yazar and A. Dunkels. Efficient Application Integration in IP-based Sensor Networks. In *Proceedings of the 1st Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)*, pages 43–48, 2009.
- [160] K. Yedavalli, B. Krishnamachari, S. Ravula, and B. Srinivasan. Ecolocation: A Sequence Based Technique for RF Localization in Wireless Sensor Networks. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 285–292, 2005.
- [161] S. Yoon, C. Veerarittiphan, and M. L. Sichitiu. Tiny-Sync: Tight Time Synchronization for Wireless Sensor Networks. *ACM Transactions on Sensor Networks*, 3(2), 2007.

- [162] J. Zhang, T. Yan, J. A. Stankovic, and S. H. Son. Thunder: Towards Practical, Zero Cost Acoustic Localization for Outdoor Wireless Sensor Networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 11(1):15–28, 2007.
- [163] Z. Zhong and T. He. Achieving Range-Free Localization Beyond Connectivity. In *Proceedings of the 7th Conference on Embedded Networked Sensor Systems (SenSys)*, pages 281–294, 2009.
- [164] T. Zhu, Z. Zhong, T. He, and Z.-l. Zhang. Exploring Link Correlation for Efficient Flooding in Wireless Sensor Networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 49–63, 2010.

Publications

The following list enumerates the publications I co-authored during my time as a Ph.D. student with the Distributed Computing Group at ETH Zurich. Our research group follows the convention of the distributed computing community to have **alphabetically** ordered authors for all publications.

1. **Sundroid: Solar Radiation Awareness with Smartphones.** Thomas Fahrni, Michael Kuhn, Philipp Sommer, Roger Wattenhofer, and Samuel Welten. In *Proceedings of the 13th International Conference on Ubiquitous Computing (UbiComp)*, Beijing, China, September 2011
2. **SpiderBat: Augmenting Wireless Sensor Networks with Distance and Angle Information.** Georg Oberholzer, Philipp Sommer, and Roger Wattenhofer. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, Chicago, IL, USA, April 2011
3. **Debugging Wireless Sensor Networks Simulations with YETI and COOJA.** Richard Huber, Philipp Sommer, and Roger Wattenhofer. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, Chicago, IL, USA, April 2011
4. **Ikarus: Large-Scale Participatory Sensing at High Altitudes.** Michael von Kaenel, Philipp Sommer, and Roger Wattenhofer. In *12th Workshop on Mobile Computing Systems and Applications (HotMobile)*, Phoenix, AZ, USA, March 2011
5. **Poster Abstract: Reliable and Energy-Efficient Bulk-Data Dissemination in Wireless Sensor Networks.** David Gugelmann, Philipp Sommer, and Roger Wattenhofer. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Zurich, Switzerland, November 2010

6. **Demo Abstract: The SpiderBat Ultrasound Positioning System.** Georg Oberholzer, Philipp Sommer, and Roger Wattenhofer. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Zurich, Switzerland, November 2010
7. **Clock Synchronization: Open Problems in Theory and Practice (invited paper).** Christoph Lenzen, Thomas Locher, Philipp Sommer, and Roger Wattenhofer. In *Proceedings of the 36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, Špindlerův Mlýn, Czech Republic, January 2010
8. **Optimal Clock Synchronization in Networks.** Christoph Lenzen, Philipp Sommer, and Roger Wattenhofer. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Berkeley, CA, USA, November 2009
9. **Demo Abstract: YETI - An Eclipse Plug-in for TinyOS 2.1.** Nicolas Burri, Roland Flury, Silvan Nellen, Benjamin Sigg, Philipp Sommer, and Roger Wattenhofer. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Berkeley, CA, USA, November 2009
10. **Towards a Zero-Configuration Wireless Sensor Network Architecture for Smart Buildings.** Lars Schor, Philipp Sommer, and Roger Wattenhofer. In *First ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings (BuildSys)*, Berkeley, CA, USA, November 2009
11. **Gradient Clock Synchronization in Wireless Sensor Networks.** Philipp Sommer and Roger Wattenhofer. In *Proceedings of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, San Francisco, CA, USA, April 2009
12. **Generic Mobility Simulation Framework (GMSF).** Rainer Baumann, Franck Legendre, and Philipp Sommer. In *1st ACM SIGMOBILE Workshop on Mobility Models*, Hong Kong, China, May 2008
13. **Symmetric Clock Synchronization in Sensor Networks.** Philipp Sommer and Roger Wattenhofer. In *ACM Workshop on Real-World Wireless Sensor Networks (REALWSN)*, Glasgow, Scotland, April 2008.